كلية الاقتصاد والتجارة زليتن

قسم الحاسوب

لغة الجافا

أعدها : عبدالناصر الغويل

(1)

2012

اساسيات لغة الجافا

المعرفات Identifiers

هي الكلمات التي يعرفها المبرمج ويستخدمها داخل البرنامج لتسمية اسماء الثوابت و اسماء المتغيرات واسماء الدوال أو الطرق (Methods) والفصائل (Classes) وغيرها **بشرط أن** :

* يتكون من الحروف الكبيرة (A-Z) او الصغير (a-z) او الارقام (0-9) او الشرطة التحتية (_) او علامة الدولار ($).

<u>**مثلا**</u> : `no` ، `number2` ، `price_4` ، `balance$` ، `salary$_2012` ، كلها اسماء معرفات صحيحة.

* ان لا يبدأ برقم وربما يلحقه أي تشكيلة مما سبق ذكره.

<u>**مثلا**</u> : `2salary` ليس اسم معرف صحيح لأنه يبدأ برقم.

* ان لا يحتوي على أي من الرموز الخاصة أو الفراغات.

<u>**مثلا**</u> : `area#2` ليس اسم معرف صحيح لأنه يحتوي على رمز خاص.

`Student Number` ليس اسم معرف صحيح لأنه يحتوي على فراغ.

* لا يسمح باستخدام الكلمات المحجوزة (*Reserved Words*) كأسماء للمعرفات.

<u>**مثلا**</u> : `class` أو `public` ليست اسماء معرفات صحيحة لأنها كلمات محجوزة.

* يفضل ان يكون اسم المعرف ذات مدلول ومعنى لما اختير له .

<u>**مثلا**</u> : `button1` أو `btn1` هو وصف واضح لاسم الزر الاول.

ملاحظة :

الجافا حساسة لحالة الاحرف (*sensitive*) ، حيث يتم معاملة الحروف الكبيرة بشكل مختلف عن الحروف الصغيرة فكلمة `number` تختلف عن كلمة `Number` .

تدريب (1)

أي من الاسماء التالية يمكن استخدامها كأسماء للمعرفات :

اسم المعرف	هل هو صحيح أو لا	في حالة (لا) اذكر السبب
`Private`	....	....
`$name`	....	....
`_Quantity`	....	....
`five_5`	....	....
`Windows 7`	....	....
`c#`	....	....
`sql`	....	....
`$_88`	....	....
`item-8`	....	....

ملاحظة :

يعتبر اسلوب جيد عند كتابة البرامج بلغة الجافا ان يبدأ كتابة اسماء المتغيرات واسماء الطرق او الدوال بحرف صغير للكلمة الاولى وكتابة حرف كبير لكل كلمة لاحقة فمثلا:

رقم الطالب **s**tudent**N**umber نلاحظ كتابة أول كلمة بحرف صغير والكلمة التالية لها بحرف كبير مع عدم ترك فراغ طبعا. أو كتابة **g**et**E**mployee**N**ame كاسم للطريقة في الحصول على اسم الموظف.

أما في حالة كتابة اسماء للفصائل (class) فيفضل أن يكون اول حرف كبير فمثلا :

اذا كان لدينا فصيل للطالب فيمكن كتابة اسم الفصيل كالتالي : **S**tudent .

أما اسماء الثوابت فيفضل كتابها كلها بحروف كبيرة مع فصل كل كلمة عن اخرى بالشرطة التحتية (_) فمثلا :

MAXIMUM_NUMBER ، ARRAY_SIZE

أنواع البيانات Data Types :

هناك نوعان من البيانات في لغة الجافا :الاول يكون معرفا في اصل اللغة (primitive type) والثاني يقوم المستخدم بتعريفه (User-defined type) .

حيث يمثل النوع الاول (primitive type) اسم يخزن فيه قيمة معينة (مثلا : قيمة ثابتة او قيمة متغير اخر او نتيجة منهج) بشرط الالتزام بشروط المعرفات سالفة الذكر.

يتم تعريف المتغيرات لتحديد نوعها ، حيث انه قبل استخدام أي متغير يجب تعريفه وتحديد نوعه كالتالي :

Variable_ type *Variable_ name* ;

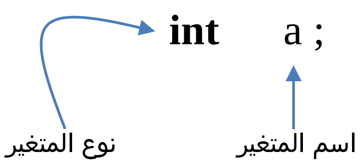
أنواع البيانات الاساسية Primitive Type:

هناك أربعة انواع من البيانات الاساسية التي تأتي في أصل لغة الجافا وهي :

1- المتغير الصحيح int :

متغير يخزن به الاعداد الصحيحة الموجبة والسالبة على ان لا يحتوي على الفاصلة العشرية.

يعرف كالتالي :



مثال لتخصيص قيم لمتغير صحيح:

a = 100 ;   a = -200 ;   a = 0;

امثلة خاطئة لتخصيص قيمة لمتغير صحيح

لوجود فاصلة عشرية   a = 100.25;

لوجود رمز خاص   a = 500$;

عند وجود عدة متغيرات صحيحة يمكن تعريفها كالتالي :

int *a* , *b* , *c* , *d* ;

كما يمكن تعريف المتغير الصحيح ومنحه قيمة مبدئية عند بداية التعريف كالتالي :

int a = 10;

هناك انواع اخرى من المتغيرات الصحيحة تختلف في السعة كالتالي :

Type	Range	Size
byte	-128..127	8-bit
short	-32768..32767	16-bit
int	-2147483648..2147483647	32-bit
long	-2^{63} .. 2^{63} - 1	64-bit

أمثلة:

```
int    a ;
long   b ;
short  c ;
byte   d ;
```

ثم التخصيص كالتالي :

```
a = 9000;
b = 9000000;
c = -452;
d = 88;
```

2- المتغيرات الحقيقة float :

هي المتغيرات التي يمكن ان تخزن كسور عشرية وتعرف كالتالي :

```
float   a;
```

ثم التخصيص كالتالي :

```
a = 9.5f ;
```

لاحظ ان القيمة المخصصة للمتغير الحقيقي تنتهي بـ(f) .

يكون مجال النوع (**float** <= 32 bits) من `-3.4x10^38 to 3.4x10^38` وبحوالي سبع خانات بعد الفاصلة الكسرية.

كما يوجد نوع كسري اخر أكبر سعة هو (**double** <= 64 bits) والذي يتراوح مجاله من `-1.8x10^308 to 1.8x10^308` وبحوالي 15 رقم بعد الفاصلة العشرية.

```
double    a;
```

ثم التخصيص كالتالي :

```
a = 9.123456789123456;
```

3- المتغيرات المنطقي boolean :

هو متغير يمكن ان يحتوي على احدى قيمتين ***True*** أو ***False*** .

مثلا :

```
boolean b ;
```

ثم في التخصيص:

```
    b = true;
    b = false;
```

4- المتغيرات الحرفية char :

هي متغيرات تحتوي على رمز واحد فقط ضمن شفرة الـ(unicode) بين علامتي التنصيص المفردة.

مثلا :

```
char   c ;
```

ثم في التخصيص:

```
    c = 'A';
    c = '#';
```

ملاحظة :

نوع السلسلة الرمزية **String** ليس نوع اساس (*Primitive type*) ضمن الجافا انما هو فصيل (*class*) له العديد من الطريق (*methods*) والتي يمكن الاستفادة منها حيث *سيتم شرحها لاحقا* ، ولكن على كل حال يمكن حاليا تعريف نوع من السلسلة الرمزية بطريقة مبسطة كالتالي :

```
String s = "Ali";
```

لاحظ ان اول حرف في **String** هو حرف كبير (لأنه اسم فصيل)

لاحظ ايضا الفرق بين قيم المتغيرات الرمزية (*char*) والتي تكون بين علامتي التنصيص المفردة وكائنات السلسلة الرمزية (*String*) والتي تكون بين علامتي التنصيص المزدوجة.

الكلمات المحجوزة Reserved Words :

هي كلمات لها معنى خاص لدى المترجم ككلمة عام public ، عندما يجد المترجم هذه الكلمة المحجوزة سيقوم بعمل ما (يجعل الفصيل او الطريقة او المتغير عاما مثلا)

امثلة:

boolean - default - this - private - return - class - switch

التعليقات Comments :

هي جمل توضيحية ليس لها تأثير على البرنامج ويمكن كتابتها في أي مكان ضمن البرنامج ، تستخدم فقط لبيان عمل البرنامج ليسهل تعديله وقرأته من قبل الاخرين. من اساليب البرمجة الجيدة هو كتابة تعليقات التوثيق ليسهل على المبرمج تعديل برنامجه بعد فترة من الزمن ، اضافة الى ان البرنامج يصبح قابل للقراءة من قبل المبرمجين الاخرين . تعليقات التوثيق في الجافا تساعد على انتاج ملفات *HTML* باستخدام ما يعرف بـ(*javadoc*) حيث من خلال هذه الوثائق يتم بيان الغرض من كل طريقة اضافة الى البارامترات والقيم المرجعة منها . سيتم شرح كيفية انتاج هذه الوثائق لاحقا.

يكتب التعليق في لغة الجافا بطريقتين :

الاولي : عندما يمتد التعليق لأكثر من سطر واحد يكتب كالتالي :

```
/*   Here is your Comment
These comments are divided
into many lines   */
```

والثانية : عند يكون التعليق ضمن سطر واحد يكتب كالتالي :

// *Maybe here is also your comment in one line*

ملاحظة :

كل جملة في لغة الجافا تنتهي بفاصلة منقوطة (**;**) .

الثوابت Constants :

هي قيم لا يمكن تغييرها اثناء تنفيذ البرنامج. وأي قيمة ثابتة نكتب امام تعريفها كلمة **final** أي تصبح قيمة نهائية غير قابلة للتغيير.

مثلا : (ط) في حساب مساحة الدائرة ، لجعلها قيمة ثابتة اثناء تنفيذ البرنامج نكتبها كالتالي :

```
final float PI = 3.14;
```

لاحظ ان اسم الثابت يكتب بحروف كبيرة وفي حال ما إن كان الثابت مكون من اكثر من كلمة واحدة يتم الفصل بينها بالشرطة التحتية فمثلا :

```
final int MAX_PRICE = 250;
```

ملاحظة :

أما عن الانواع التي يعرفها المستخدم فهي عبارة عن الفصائل (*classes*) والتي يمكن معاملتها على انها متغيرات كما سيتم شرحه لاحقا .

الهيكل العام لمشروع جافا :

عادة يتكون مشروع الجافا من عدة فصائل classes كل فصيلة ترتبط بالأخريات لتكون في النهائية ما يعرف بمشروع الجافا Project ، لذلك سنقوم بدراسة الهيكل العام للفصيلة اولا حيث ان الفصيلة هي احدى المفاهيم الاساسية في البرمجة ذات التوجيه الغرضي او البرمجة بالأهداف *Object-Oriented Programming* (OOP) ، في هذه البرمجة يتم تمثيل كل فصيل كما هي في واقع الحياة فمثلا : فصيلة **الطالب** فلكل طالب عدد من الصفات (*attributes*) تبين حالته (*state*) مثل رقم الطالب اسم الطالب ، الدرجة ، الطول ، اللون ... الخ ، كما ان لكل كائن (طالب) مجموعة من المناهج (*methods*) تحدد تصرفاته او سلوكه (*behaviours*) فمثلا سيكون هناك منهج او دالة لإضافة درجات للطالب او دالة لتعديل احدى بياناته مثلا الاسم او اللون او دالة لطباعة تقرير درجاته ...الخ.

الهيكل العام للفصيلة :

كلمات محجوزة — اسم الفصيل

```
public class FirstProgram
{

      Fields / attributes / variables
      ………………….
      ………………….

      public FirstProgram ()
      {
                 ……………………
      }

      Methods
      ……………………
      ……………………
      ……………………
}
```

- Fields / attributes / variables: اسماء الحقول او الصفات او المتغيرات
- public FirstProgram (): الباني constructor
- Methods: المناهج او الدوال

اسم الفصيل **Public class *FirstProgram*:**

أول سطر يحدد فيه اسم الفصيلة ، حيث ان :

كلمة **Public** تعني ان هذه الفصيلة عامة ومتاحة الاستخدام من قبل الفصائل الاخرى.

كلمة **class** هي كلمة محجوزة تدل على الفصيلة.

كلمة *FirstProgram* هو اسم للفصيلة يختاره المبرمج ويفضل ان يكون ذات معنى دال على الفصيلة ويخضع لشروط المعرفات السالف ذكرها.

ملاحظة :

* يجب أن يتم تخزين هذه الفصيلة في ملف وبلاحقة (Java.) بنفس اسم الفصيل الوارد في الجملة السابقة أي يخزن كالتالي : FirstProgram.java.

* عند اختيار اسم للفصيلة اعتيد ان يكون أول حرف كبير وكل أول حرف ضمن الكلمات اللاحقة في الاسم كبيرة ايضا اما باقية الحروف فتكون صغيرة فمثلا : **F**irst**P**rogram

الحقول **Fields:**

يتم هنا تعريف حقول الفصيل سواء ان كان احدى الانواع السالفة الذكر (مثلا صحيح int ، او كسري double ، أو رمزي char ...الخ) أو تعريف لكائنات من فصائل اخرى (مثلا String). تعرف احيانا هذه الحقول بالصفات (*attributes*) أي تحدد صفات الكائن ، او المتغيرات (variables) . عادة ما تكون حقول الفصيلة محمية وذلك بأن تكون اما خاصة بالفصيلة (*private*) او محمية (*protected*) لتكون ممكنة الوصول من قبل الفصائل الابناء او الفصائل الموجودة بنفس الحزمة ، ويتم الوصول اليها عن طريق مناهج *methods* او دوال التخصيص (***set***) او دوال الحصول (***get***) على قيم الحقول وهذا ما يعرف في البرمجة ذات التوجيه الغرضي (*OOP*) بالكبسلة او التغليف (*Encapsulation*). سيتم شرح المحددات او المعرفات (*modifiers*) مثل (العام *public* ، والخاص *private* ، المحمي *protected* .. الخ) لاحقا.

امثلة لتعريف بعض الحقول:

```
private double maximumPrice;
private String studentName;
private int age;
Student std1 = new Student(100, "Ali") ;
```

حيث يتم كتابة المحدد (درجة الوصول الى الحقل) مثلا خاص **private**، ثم نوع الحقل (**int** أو **String** مثلا)، ثم اسم الحقل ويخضع لشروط تسمية المعرفات في الجافا ثم الفاصلة المنقوطة.

ملاحظة :

* عند اختيار اسم للحقل اعديت ان يكون أول حرف صغير وكل أول حرف ضمن الكلمات اللاحقة في الاسم كبير فمثلا: student**N**ame .

* عند عدم استخدام محدد مثل (*private* أو *public*) فهذا يعني ان امكانية الوصول ستكون افتراضية ***default*** أي يمكن الوصول اليها من نفس فصائل الحزمة فقط كما سيتم شرحه لاحقا.

البانى : Constructor :

```
public FirstProgram()
 {
      ......................................
 }
```

من احدى مفاهيم البرمجة الهدفية (OOP) هو وجود دالة تسمى الباني (*Constructor*) تنفيذ بشكل تلقائي عند تكوين كائن من الفصيل وبدون أي استدعاء ظاهري واضح ، حيث أنها عبارة عن دالة او منهج *method* مثل أي منهج اخر عادة ما يستخدم لإعطاء قيم ابتدائية لحقول الفصيلة وعادة ما يعرف بالباني الافتراضي (*default*). عادة ما يكون سماحية الوصول الى هذا الباني عاما (***public***) ويأخذ اسما هو نفس اسم الفصيل ففي مثالنا هو (FirstProgram) وبدون ترجيع أي قيمة ، ويتم حصر جميع جمل الباني بين قوسي الفئة {   } . كما أن الباني ليس اجباري ان يكون موجودا بالفصيل ، ايضا يمكن ان يكون هناك اكثر من باني واحد ضمن الفصيل متفقة في اسم المنهج *method* وتختلف في عدد ونوع البارامترات وهي ما يسمى بالتحميل الزائد للمنهج (*Overloading*) كما سيوضح لاحقا.

المناهج Methods :

ان البرامج الفعلية عادة ما تكوون كبيرة الحجم ونادرا ما تجد برنامج كتلة واحدة ، حيث يتم تقسيم البرنامج الى برامج صغيرة تعرف بالمناهج *methods* او قد تسمى احيانا دوال *functions* او اجراءات *procedures* او برامج فرعية *subroutines* . حيث كل منهج يقوم بوظيفة معينة يمكن استدعائه حين الحاجة اليه كما يمكن استدعائه لأكثر من مرة ليوفر علينا اعادة برمجته من جديد. اضافة الى ان هذه المناهج يمكن استخدامها من قبل الاخرين ودون الحاجة الى معرفة تفاصيل البرمجة فيها ، فمثلا : يمكننا استدعها العديد من الدوال الرياضية الجاهزة دون الحاجة الى معرفة تفاصيل برمجتها ، فقط نستدعيها وكل ما نحتاجه هو ذكر اسمها وتمرير بعض المعاملات لها.

مثلا : اذا كانت كان لدينا منهج لطباعة جملة ما كالتالي :

```
Public void printFirstStatement()
{
    System.out.print ("This is the First Line.");
}
```

ضمن هذه السطور نجد أنه تم تحديد نوع المحدد *modifier* للمنهج أي هل هذا المنهج سيكون عام (***public***) او خاصا (*private*) ... الخ . ثم بعد ذلك كتب نوع القيمة التي سيرجعها المنهج اما صحيحة (*int*) او كسرية (*double*) او سلسلة حروف (*String*) او كائن (*Student* مثلا) او لا يرجع شيء (***void***) كما في مثالنا هذا أي ان المنهج سوف يقوم بعمل ما وبدون ارجاع أي قيمة. ثم يلي ذلك اسم المنهج وهو هنا **p**rint**F**irst**S**tatement( ) ، لاحظ أن اسم المنهج يتبعه قوسين قد يستقبل من خلالها بارامترات تمرر للمنهج كما سيوضح بعد قليل. عادة ما يتم تسميه المنهج باسم يبدأ بفعل اول حرفه صغير ثم اول حرف من كل كلمة تالية كبير.

تحصر كل جمل المنهج بقوسي فئة { } . كما أن كل جملة ضمن لغة الجافا يجب ان تنتهي بفاصلة منقوطة .

هذا المنهج يمكن استدعائه من داخل نفس الفصيل وذلك بذكر اسمه متبوعا بقوسين ( ) ثم الفاصلة المنقوطة كالتالي :

```
printFirstStatement();
```

مما سيؤدي على تنفذ جمل المنهج وهي هنا طباعة العبارة :

```
This is the First Line.
```

اما استدعائه من خارج الفصيل فيتم ذلك بذكر اسم الكائن (*object name*) والذي تم تكوينه من الفصيل ، ثم نقطة ثم اسم المنهج متبوعا بقوسين وفاصلة منقوطة كالتالي :

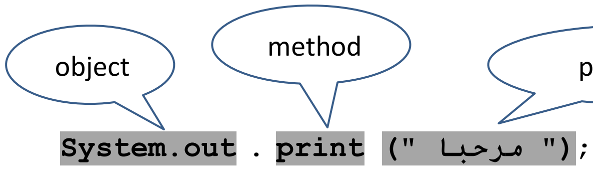
```
objectName.printFirstStatement();
```

والذي يؤدي الى ذات النتيجة وهي طباعة نفس العبارة :

```
This is the First Line.
```

منهج الطباعة print() و println():

يستخدم منهج الطباعة print() لطباعة المخرجات على الشاشة مستخدمين الفصيلة المسمى **S**ystem ، حيث أن out هو كائن (object) من الفصيلة System يمثل نافذة المخرجات .



فللطباعة على الشاشة نكتب اسم الفصيل System متبوعة بنقطة ثم out متبوعة بنقطة ثم منهج الطباعة print() ونضع ما نريد طباعته بين القوسين كبارامترات .

فاذا كان لدينا جملتي الطباعة التاليتين :

```
System.out.print ("This is the First Line.");
System.out.print ("This is the Second Line.");
```

فان نتيجة الطباعة ستكون كالتالي :

```
This is the First Line.This is the Second Line.
```

لاحظ ان طباعة الجملتين كانتا على نفس السطر رغم استخدام جملتي طباعة منفصلتين.

يمكن استخدام منهج الطباعة println() والذي سيؤدي الى طباعة الجملة الاولى على السطر الاول ثم انزال المؤشر الى السطر التالي مؤثرا على جملة الطباعة التالية لتكون على سطر جديد. فمثلا :

```
System.out.println ("This is the First Line.");
System.out.print ("This is the Second Line.");
```

حيث ان الطباعة الان ستكون كالتالي :

```
This is the First Line.
This is the Second Line.
```

يمكن طباعة السلاسل الرمزية كما سبق ذ1كره او قيم لاحدي الانواع الاساسية (مثل الاعداد الصحيحة او الكسرية او الرمزية ...الخ) او نتيجة تعبير حسابي او منطقي :

```
System.out.println ( 50 );

System.out.println ( 50 + 100);        ===(result)=== >>    150
```

كما يمكن طباعة قيم المتغيرات كالتالي :

```
a = 200;
System.out.println ( a );              ===(result)=== >>   200
```

ايضا يمكن طباعة كائن *object* بعد اعادة برمجة المنهج `toString()` والموروث من الفصيل الاب (**Object**) كما سيأتي لاحقا ضمن فيما يعرف بـ Overriding.

يمكن طباعة اكثر من متغير او سلسلة ضمن جملة طباعة واحدة على ان تفصل بينها بعلامة (+) حيث تقوم علامة (+) بربط (*Concatenate*) السلاسل ببعضها كما يلي:

```
System.out.println ( "Hello" + "Ali" + "today" );
```

فتكون نتيجة الطباعة كالتالي :

```
HelloAlitoday
```

لاحظ عدم وجود فراغ في نتيجة الطباعة بين الكلمات الثلاثة، فلترك فراغ نحشر سلسلة حرفية فارغة " " بين السلاسل كالتالي :

```
System.out.println ( "Hello" + "  " + "Ali" + "  " + "today");
```

فتكون نتيجة الطباعة كالتالي :

```
Hello   Ali   today
```

أما عند استخدام سلسلة رمزية مع قيمة عددية فسيتم تحويل القيمة العددية الى سلسلة رمزية ثم ربطها مع السلسلة الرمزية الاولى فمثلا :

```
System.out.println ( "Hello" + 50 );
```

حيث النتيجة ستكون كالتالي :

```
Hello50
```

وكذلك الحال عندما تكون القيمة العددية هي الاولى والسلسلة هي الثانية :

```
System.out.println ( 50 + "Hello" );
```

فان النتيجة ستكون كالتالي :

```
50Hello
```

تدريب (2)

ما نتيجة جمل الطباعة التالية :

<u>النتيجة</u>

```
System.out.println ( 120 + "Hello");            ................................................
System.out.println ( 120 + 40 + "Hello"); ................................................
System.out.println ( 120+"Hello" + 60);     ................................................
System.out.println ( "Hello" + 120 + 40);  ................................................
System.out.println ( "Hello" + (120 + 40) );  ......................................
```

ذكرنا فيما سبق اننا عندما نريد ان نطبع عدة جمل كل جملة على سطر منفصل نستخدم الدالة println() ، يمكن عوضا عن ذلك استخـــدام ("\n") احدى حرفيــــات الهــــروب (*Escaping Literals*) على ان تكون محصورة بين علامتي التنصيص المفردة او المزدوجة وهي تقريبا نفس رموز الهروب الموجود في لغة C , C++ حيث يمكن استخدام "\n" للقفز الى سطر جديد (نفس عمل مفتاح Enter) ، والرمز '\t' لترك سبع مسافات عمودية (نفس عمل مفتاح tab) ، والرمز " '\ " لطبعة علامة التنصيص المفردة ، والرمز " "\ " لطباعة علامة التنصيص المزدوجة ، والرمز "\\"لطباعة رمز الهروب نفسه (\) ... وهكذا .

ملاحظة :

* يمكن طباعة الرموز بالنظام الثماني باستخدام رمز الهروب "\ddd" او بالنظام السادس عشري ضمن شفرة الـ(unicode) باستخدام رمز الهروب "\uxxxx".

* سنقوم باستخدام برنامج BlueJ كبيئة لتنفيذ برامج الجافا. حيث سنستخدم الرمز '\u000C' بالنظام الساس عشر ضمن شفرة الـ (nuicode) لتنظيف الشاشة كالتالي :

```
System.out.println ('\u000C');
```

منهج الطباعة ()printf :

لكون لغة الجافا مبنية اساسا من لغة السي ، فيمكن أن نستخدم منهج الطباعة ()printf بدلا من المنهج ()print لطباعة المخرجات كحال استخدامها تماما في لغة السي ، فمثلا :

```
System.out.printf ( "Hello" );
```

فتطبع كلمة :

```
Hello
```

لاحظ أنه يفصل بين القيم والسلاسل ضمن دالة الطباعة ()printf بالفاصلة (,) وليس علامة (+) كالتالي :

```
System.out.printf ( "Hello" , "Ali");
```

ايضا يمكن استخدامها بمواصفات التحويل لتنسيق ظهور الاعداد كالأعداد الحقيقة مثلا كالتالي :

```
System.out.printf ( "Price = %6.3f" , 12.5);
```

حيث تبدأ بعلامة % متبوعة برمز من رموز التحويل التالية ذكرها ، فالنتيجة تكون في المثال السابق هو حجز ستة خانات : واحدة للنقطة العشرية وثلاث خانات كسرية والباقي للجزء الصحيح ليكون تنسيق المخرجات كالتالي :

```
Price = 12.500
```

امثلة متنوعة :

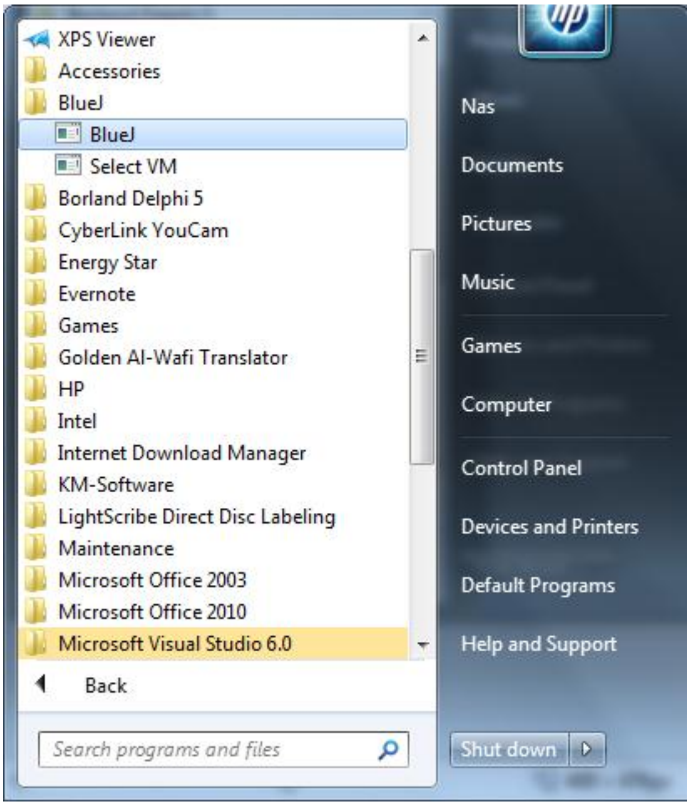
النوع	الرمز	مثال	النتيجة
الاعداد الصحيحة	%d	`System.out.printf ( "Hello %d" , 10);`	Hello 10
الاعداد الكسرية	%f	`System.out.printf ( "Hello %7.2f" , 52.6);`	Hello 52.60
السلاسل الرمزية	%s	`System.out.printf ( "Hello %s" , "Ali");`	Hello Ali
الحروف	%c %C	`System.out.printf ( "Hello %c" , 'r');`	Hello r
عدد صحيح بالنظام الثماني	%o	`System.out.printf ( "Hello %o" , 10);`	Hello 12
عدد صحيح بالنظام السادس عشر	%x %X	`System.out.printf ( "Hello %x" , 10);`	Hello a
طباعة %	%%	`System.out.printf ( "Hello %%");`	Hello %

<u>المثال الاول :</u>

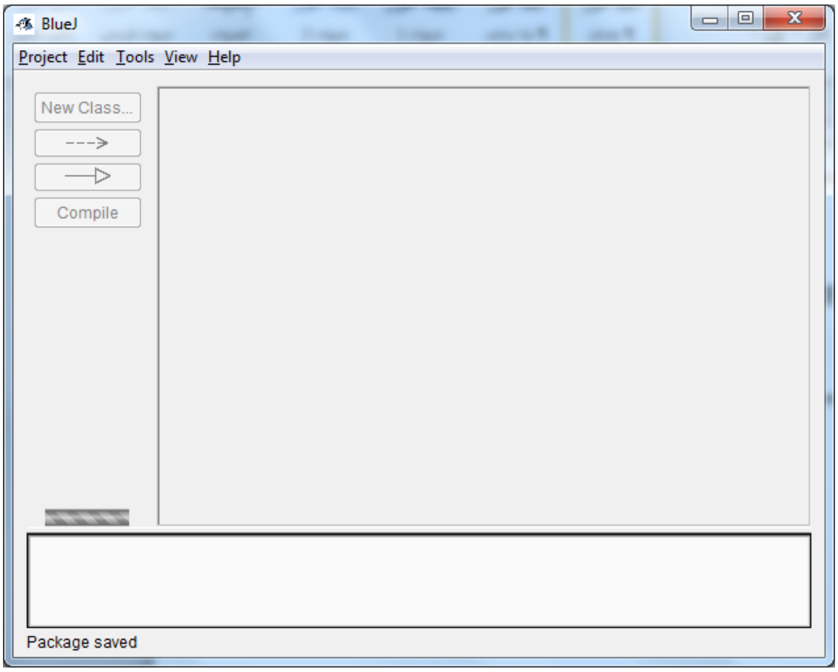
اكتب برنامج بلغة الجافا لبناء فصيلة للطالب (*Student*) تحتوي على حقل واحد فقط هو اسم الطالب (*studentName*) على ان يتم تطبيق مفاهيم البرمجة الهدفية (*OOP*) ، بمعنى تطبيق مفهوم الكبسلة أي : عدم الوصول الى الحقول مباشرة واستخدام المناهج المناسبة للحصول على قيم الحقول وتغييرها ، ايضا برمجة الباني لإعطاء قيم مبدئية للحقول.

<u>أولا</u> سوف نستخدم برنامج ***BlueJ*** كبيئة لتنفيذ برامجنا بلغة الجافا لسهولة ووضوح تكوين ملفات الجافا بها ، كما لا يخفى علينا وجود العديد من البيئات الاخرى مثل Eclipse و NetBeans والتي يمكن استخدامها لتكوين برامج الجافا وخاصة لتكوين الواجهات الرسومية. ايضا يمكن تكوين برامج الجافا في أي محرر نصوص مثل المفكرة في الويندوز ومن ثم اجراء عملية ترجمة وتنفيذ للبرنامج كما سيوضح لاحقا.

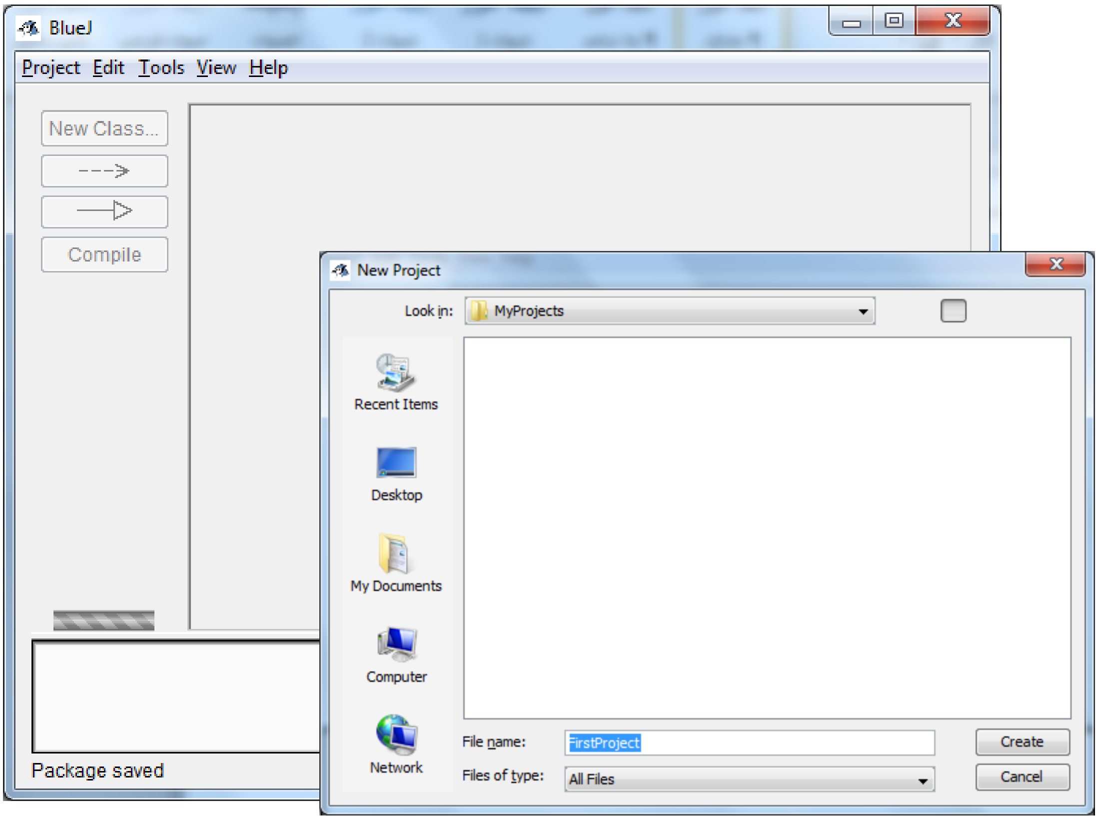
- من قائمة ابدا انتقل الى كافة البرامج (*All Programs*) ثم اختار (*BlueJ*) ثم انقر (*BlueJ*) او قم بالنقر عليه من على سطح المكتب ان كنت قد كونت له اختصار اثناء عملية التنصيب :



ستظهر لك نافذة الـ(BlueJ) كالتالي :



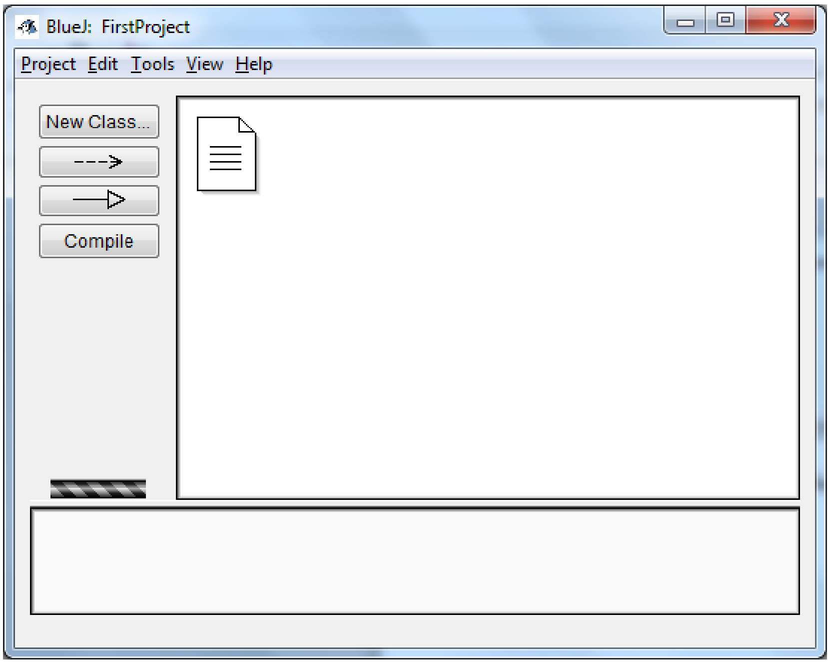
➢ من قائمة المشروع (Project) اختار مشروع جديد (... New Project) ثم حدد المسار الذي ستخزن فيه مشروعك وليكن ضمن مجلد على الـ(:F) اسمه (*MyProjects*) لتخزن به كل مشاريعك. اختر اسما لمشروعك الاول وليكن (*FirstProject*) كما موضح اسفله :



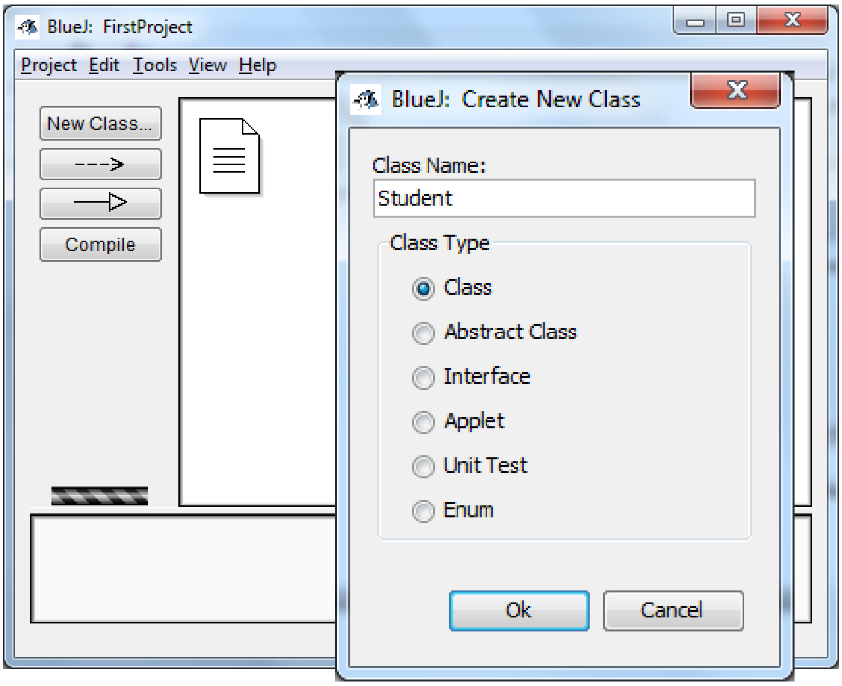
لاحظ أن مشروعك الآن ضمن مجلد فرعي في المسار التالي :

F:\MyProjects\FirtProject\ ....

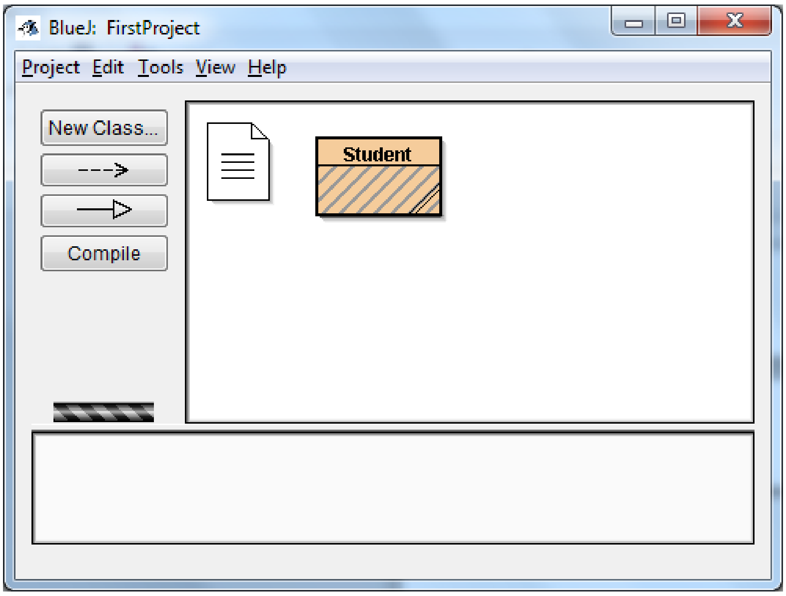
وان جميع الفصائل التي سيتم تكوينها ضمن هذا المشروع سيتم تخزينها ضمن هذا المجلد الفرعي بامتداد (.java)



- انقر على زر تكوين فصيلة جديدة (New Class ...) ، سيظهر مربع حوار ، ادخل بها اسم الفصيل الجديد وليكن (Student) (**تذكر** : اول حرف كبير) ثم انقر على (Ok) :



سيظهر الان شكل الفصيل الجديد :



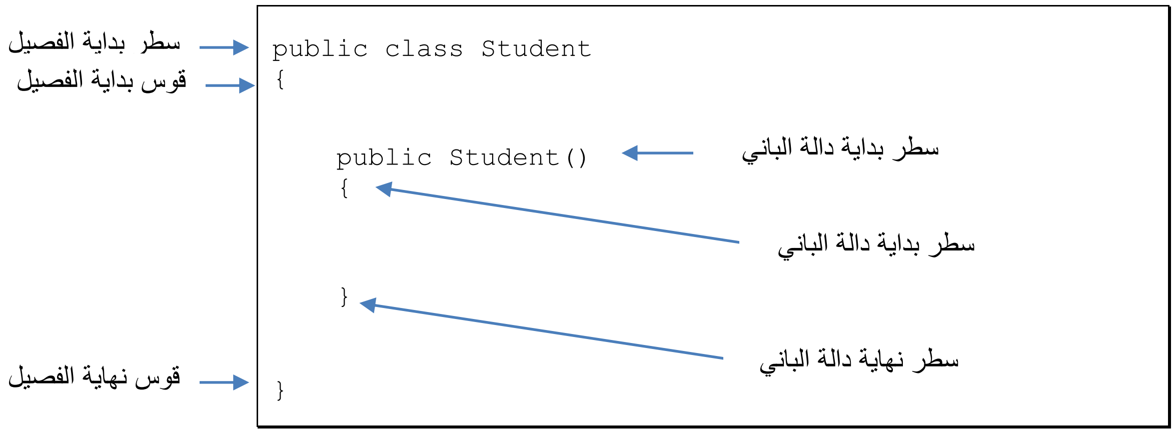
➢ انقر الان نقرا مزدوجا على الفصيل الجديد (*Student*) لتظهر لك البرمجة كالتالي :

```
/**
 * Write a description of class Student here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Student
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class Student
     */
    public Student()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     *
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

هذه البرمجة موجودة اصلا كمثال ، لذلك سنقوم دائما بحذف معظمها ليبقى شكل الفصيل كالتالي :



لاحظ أن هذا الفصيل هو عام (*public*) وان كلمة **class** هي كلمة محجوزة يجب كتابتها كما هي ، ثم يأتي بعدها اسم الفصيل و هو نفس الاسم الذي ادخلته منذ قيل (*Student*) .

كل فصيل يجب ان يبدا بقوس فئة مفتوح وينتهي بقوس فئة مغلق.

أما لبرمجة الباني : نلاحظ أن دالة الباني عادة ما تكون عامة ، أي يمكن الوصول اليه من خارج الفصيل ثم تلحقه اسم الفصيل مباشرة ثم يليه القوسين. ايضا دالة الباني يجب ان تبدأ بقوس فئة مفتوح وتنتهي بقوس فئة مغلق.

- الخطوة التالية هي اضافة حقول الفصيل وعندنا فقط حقل واحد هو اسم الطالب (*studentName*) نوعه سلسلة حروف (String) ، ولتحقيق مفهوم الكبسلة عادة ما تكون الحقول خاصة (private) فنعرف الحقل كالتالي قبل دالة الباني :

```
private    Type    fieldName;
```

```
private String studentName;
```

ليكون البرنامج كالتالي :

```
public class Student
{

    private String studentName;

    public Student()
    {

    }

}
```

- ضمن دالة الباني سنقوم بوضع قيم افتراضية مبدئية للحقول و هي عادة ما تكون اصفارا للقيم العددية و قيمة السلسلة فارغة لحقول السلاسل كالتالي :

```
studentName = "";
```

ليكون البرنامج كالتالي :

```
public class Student
{
    private String studentName;
    public Student()
    {
        studentName = "";
    }
}
```

- اما عن كيفية الوصول الى الحقل واسترجاع قيمته سنقوم ببرمجة دالة او منهج يسمى getStudentName() يأتي موقعها بعد دالة الباني وتكون مهمتها فقط استرجاع قيمة حقل اسم الطالب *studentName* كالتالي :

```
public String getStudentName()
{
    return (studentName);
}
```

هذه الدالة تكون عامة الوصول (*public*) ثم نحدد نوع القيمة المرجعة من الدالة وهي هنا اسم الطالب ونوعها String ، ثم بعد ذلك نكتب اسم المنهج او الدالة ويخضع لشروط كتابة المعرفات في الجافا السالف ذكرها ، يلي ذلك قوسين ( ).

تبدأ كل دالة او منهج بقوس فئة مفتوح { وينتهي بقوس فئة مغلق } .

كون هذه الدالة سترجع قيمة من نوع ما فيجب كتابة كلمة **return** بمعني (رجع) ثم تحدد القيمة المراد ترجيعها وهي هنا اسم الطالب *studentName* ثم تنتهي الجملة بالفاصلة المنقوطة.

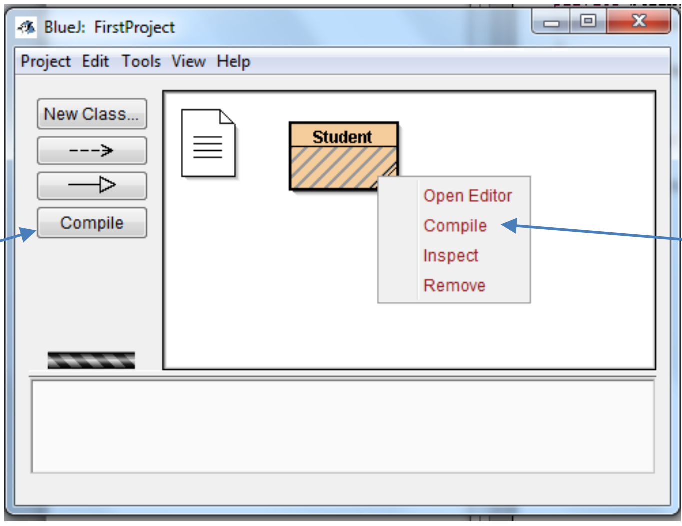
```
public class Student
{
    private String studentName;
    public Student()
    {
        studentName = "";
    }
    public String getStudentName()
    {
       return (studentName);
    }
}
```

- الان سنقوم بترجمة البرنامج واكتشاف الاخطاء ان وجدت وتصحيحها .

لاحظ أن الجافا مثل لغة السي حساسة لحالة الاحرف فهناك فرق بين الحروف الكبيرة والحروف الصغيرة ، لذلك يجب التقيد بنفس الاسم المكتوب كأول ظهور له في البرنامج تماما لتلافي الاخطاء.

لترجمة البرنامج اما ان تنقر على زر الترجمة (Compile) الموجود امامك على الشاشة ، او تنقر بالزر الايمن للفأرة وتختار (Compile) من القائمة المنسدلة كما بالشكل :

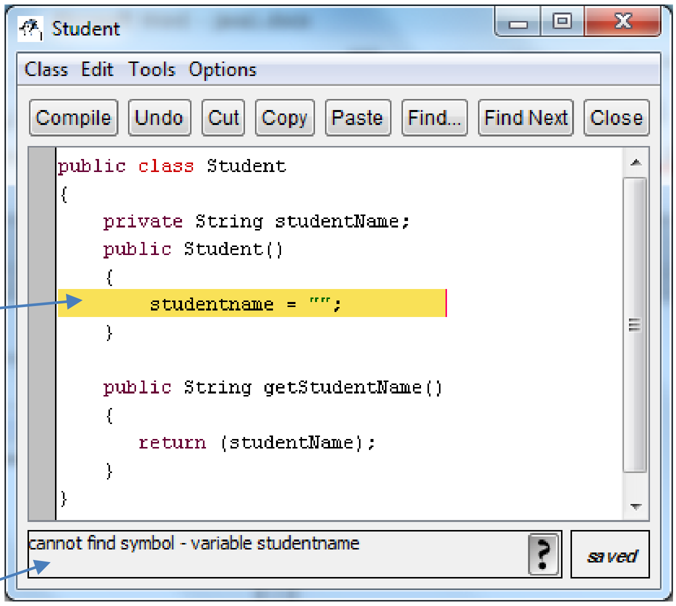
انقر على هذا الزر لترجمة البرنامج



أو تختار compile من هذه القائمة لترجمة البرنامج

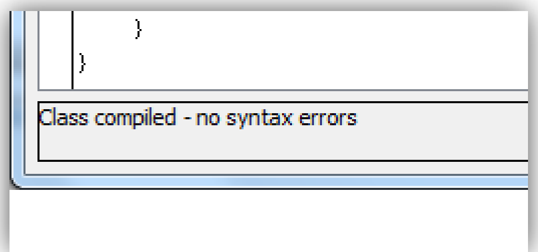
في حالة وجود اخطاء سيظهر لك شريط اصفر يحدد مكان وجود الخطأ وتظهر رسالة بأسفل نافذة البرنامج تبلغك عن ماهية الخطأ كالتالي:

مكان وجود الخطأ

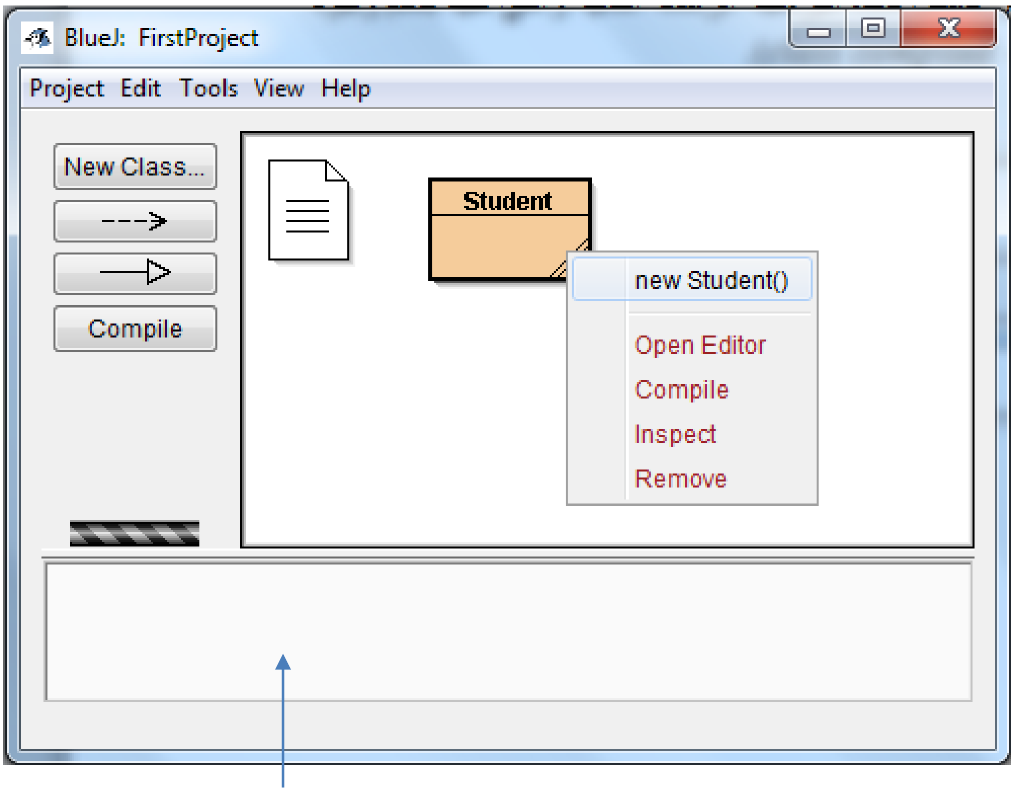


رسالة تفيد عن نوع الخطأ

بعد تصحيح الاخطاء يجب ان تظهر بأسفل نافذة البرمجة الرسالة التالية والتي تفيد بعدم وجود أخطاء لغوية (*class compiled - no syntax error*).

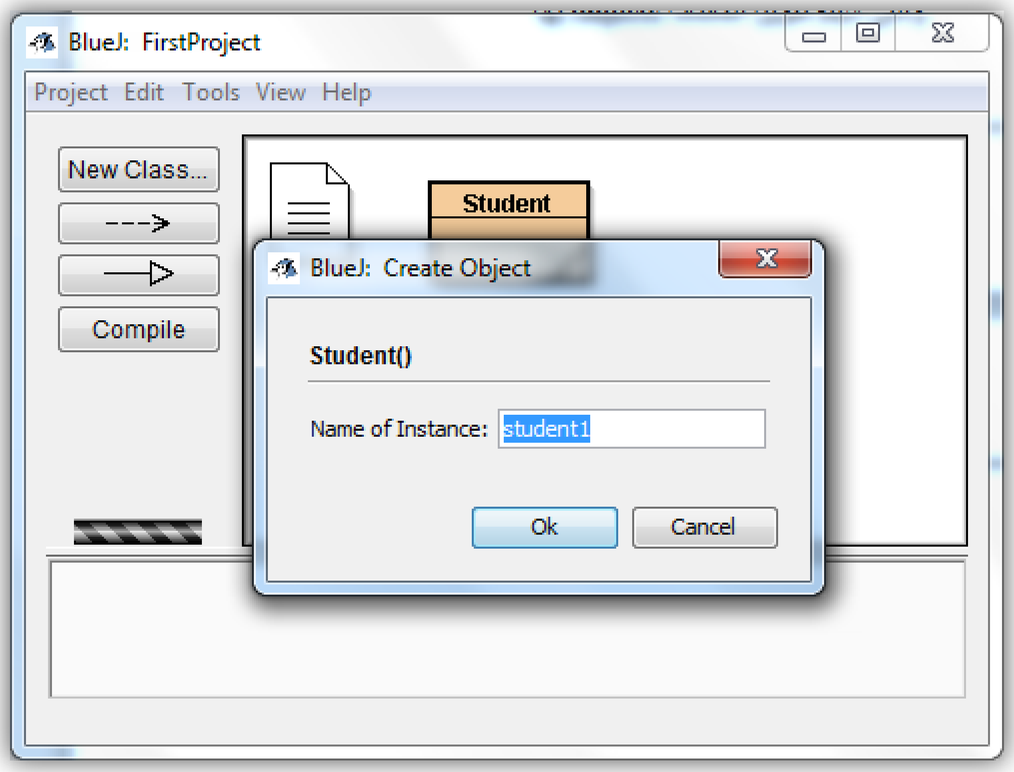


➢ الان سنقوم بتكوين كائن او هدف (**object**) من الفصيل الذي برمجناه قبل قليل وذلك بالنقر بالزر الايمن على فصيل الطالب *Student* ثم اختار *new Student()* (أي تقوم باستدعاء لدالة الباني)

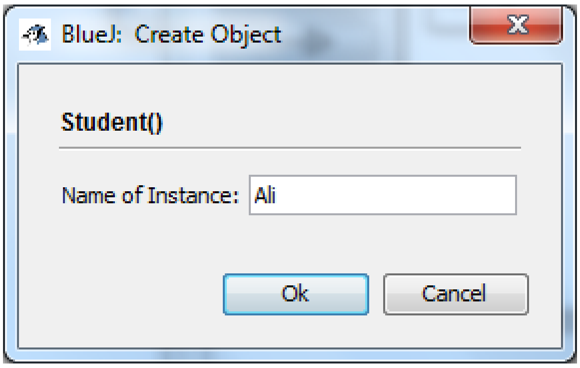


هذه المساحة تسمى بطاولة او منصة العمل workbench والتي سيتم تكوين الكائنات objects بها

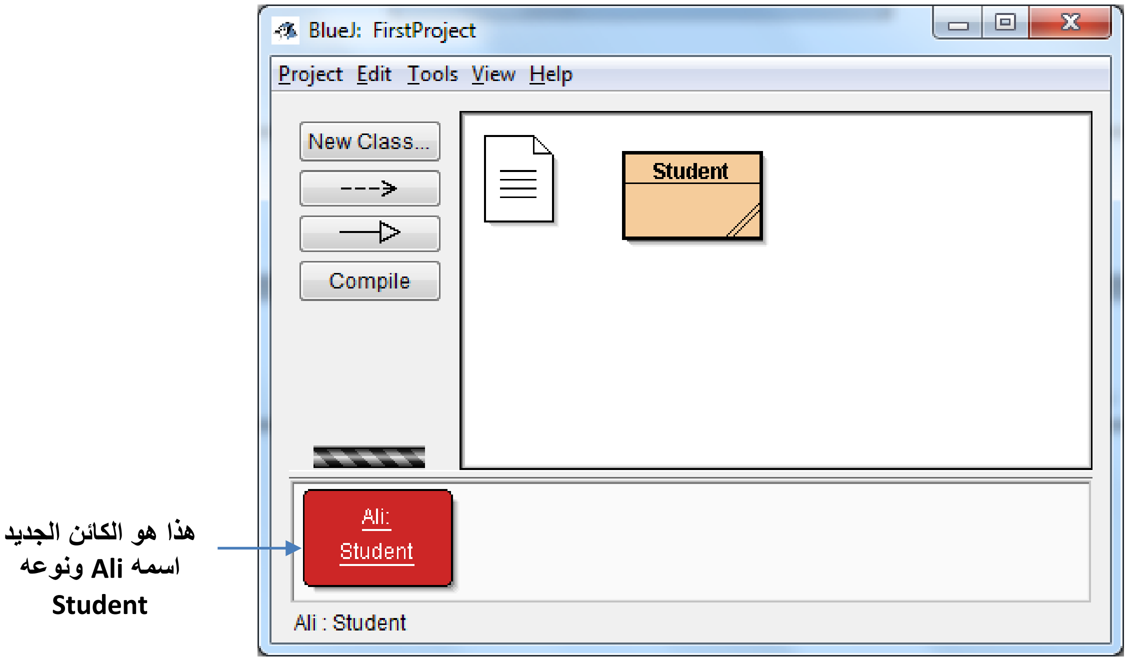
سيظهر مربع حوار يطلب منك ادخال اسم للكائن الجديد (Object) و احيانا يمسى بالمثيل (instance) أي انه مثيل عن فصيل الطالب Student ، يمكنك ان تبقي القيمة الافتراضية وهي student1 أو ان تدخل اسم للكائن يكون اكثر معنى ويدل على الطالب شرط ان يخضع لشروط تسمية المعرفات في الجافا.



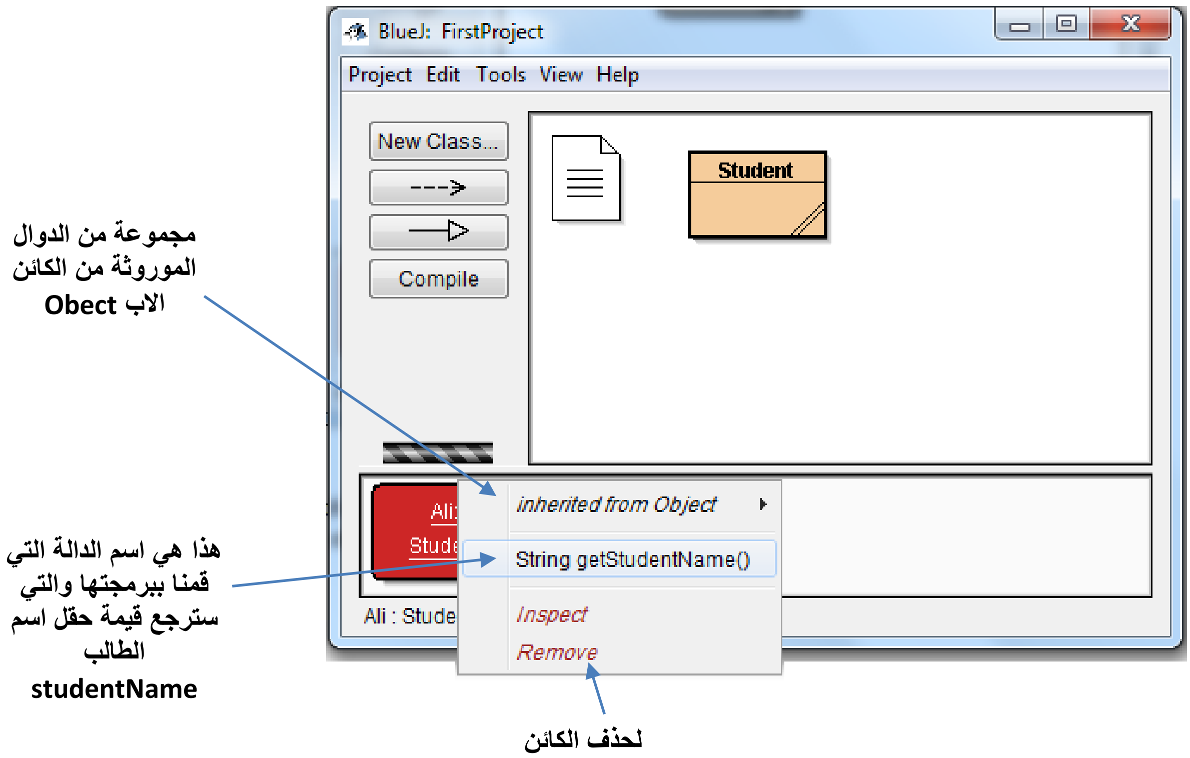
ممكن ان نختار اسما وليكن Ali كاسم للكائن ثم انقر OK كالتالي :



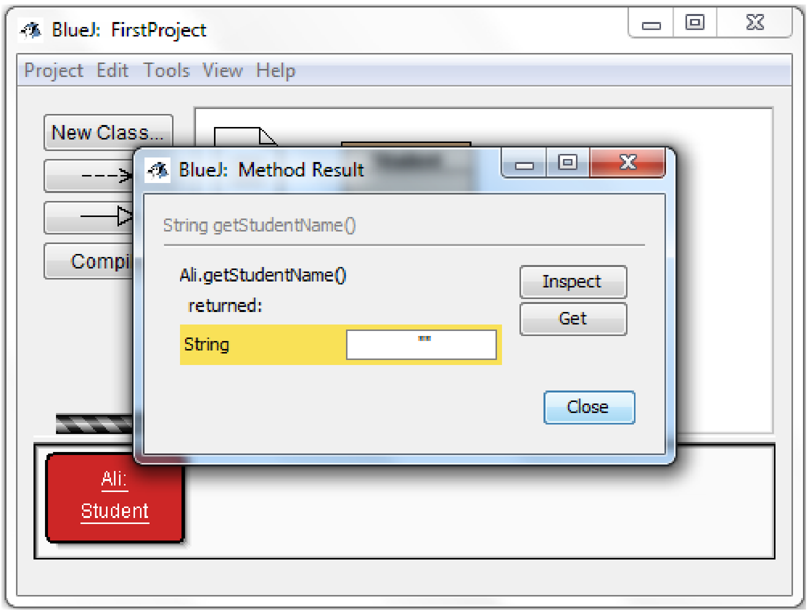
تلاحظ ظهور كائن جديد اسم Ali ونوعه Student باللون الاحمر ضمن منطقة منصة العمل كما يلي :



وبالنقر بالزر الايمن على الكائن الجديد *Ali* نشاهد قائمة بأسماء الدوال **العامة** وهي هنا فقط دالة واحدة اسمها *getStudentName()* وترجع قيمة من نوع السلسة الرمزية *String* والتي قمنا ببرمجتها قبل قليل.



من القائمة المنسدلة : وعند اختيارنا لدالة استرجاع اسم الطالب *getStudentName()* فإن النتيجة ستكون كالتالي :



لاحظ ان اسم الطالب فارغ " " وذلك لأننا ضمن دالة الباني قمنا بوضع قيمة فارغة لاسم الطالب وان دالة الباني تم استدعائها بشكل تلقائي عند تكوين الكائن *Ali* .

ولكن .... كيف يمكننا تغيير اسم الطالب ؟؟؟

سنقوم ببرمجة دالة اخرى يأتي موقعها بعد دالة *getStudentName()* تكون عامة لكي تظهر ضمن القائمة المنسدلة للكائن ووظيفتها هي استقبال سلسلة رمزية وتخزينها في حقل اسم الطالب *studentName* .

عادة ما يتم تسمية دوال تغيير قيم الحقول بالـ(set) متبوعة باسم الحقل ، رغم انه يمكنك تحديد أي اسم ترغب به بدلا من ذلك. هذه الدالة سوف لن ترجع قيمة *void* فتكون البرمجة كالتالي :

```
public void setStudentName(String n)
{
    studentName = n ;
}
```

ان دالة تغيير الحقل هي دالة عامة *public* وهي لن ترجع قيمة لان مهمتها هي استقبال قيمة ما وتخزينها في احدى الحقول لذلك استخدمنا كلمة *void* أي سوف لن ترجع شيء ، ثم يأتي بعد ذلك اسم الدالة وهي هنا *setStudentName()* متبوعة بقوسين بينها القيمة المراد استقبالها (بين القوسين يحدد نوع القيمة المراد استقبالها اولا وهي هنا String واسم للمتغير الذي سيستقبلها وهو هنا n).

القيمة التي تم استقبالها ستخزن ضمن حقل اسم الطالب كالتالي :

```
studentName = n ;
```

الان سيكون الكود كاملا هو :

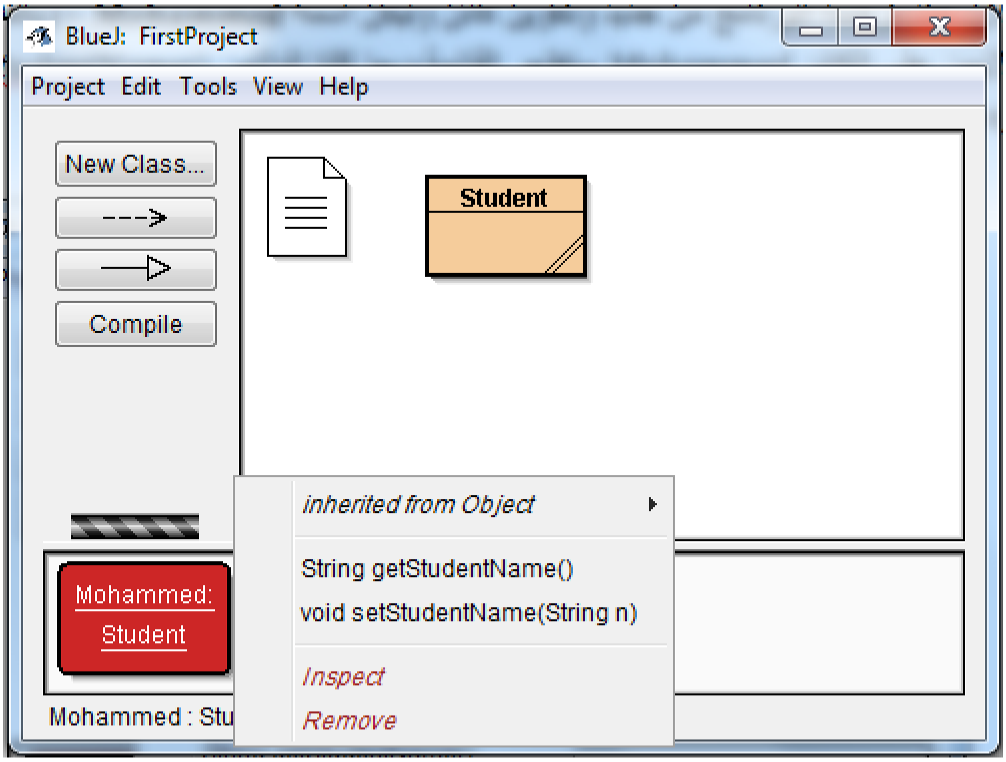
```
public class Student
{
    private String studentName;

    public Student()
    {
        studentName = "";
    }

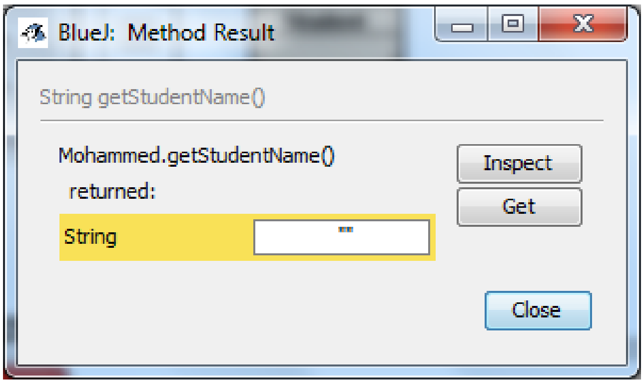
    public String getStudentName()
    {
       return (studentName);
    }

    public void setStudentName(String n)
    {
       studentName = n ;
    }
}
```

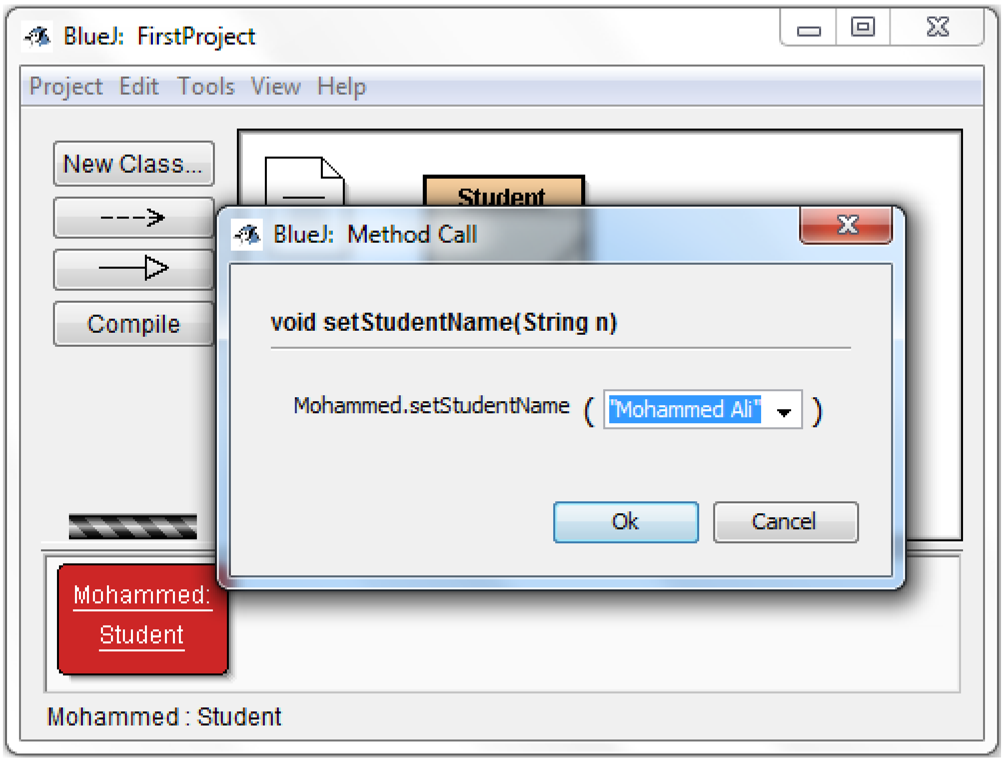
الآن عند ترجمة البرنامج من جديد وتكوين كائن وليكن اسمه ***Mohammed*** ، وبالنقر بالزر الايمن على الكائن *Mohammed* ستظهر القائمة وبها كلتا الدالتين *getStudentName()* و *setStudentName(String n)* :



اذا اخترنا دالة ترجيع الاسم *getStudentName()* فان النتيجة ستكون فارغة لاننا لم نضع بها قيمة حتى الان كالتالي :

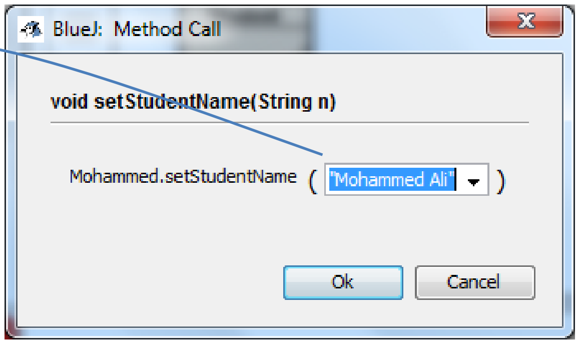


وعند اختيارنا لدالة تعديل الاسم *setStudentName(String n)* سيظهر لنا مربع الحوار التالي طالبا منا ادخال قيمة سلسلة رمزية (اسما) ليخزنه في المتغير n ولندخل الاسم "Mohammed Ali" بين علامتي التنصيص المزدوجة كالتالي :

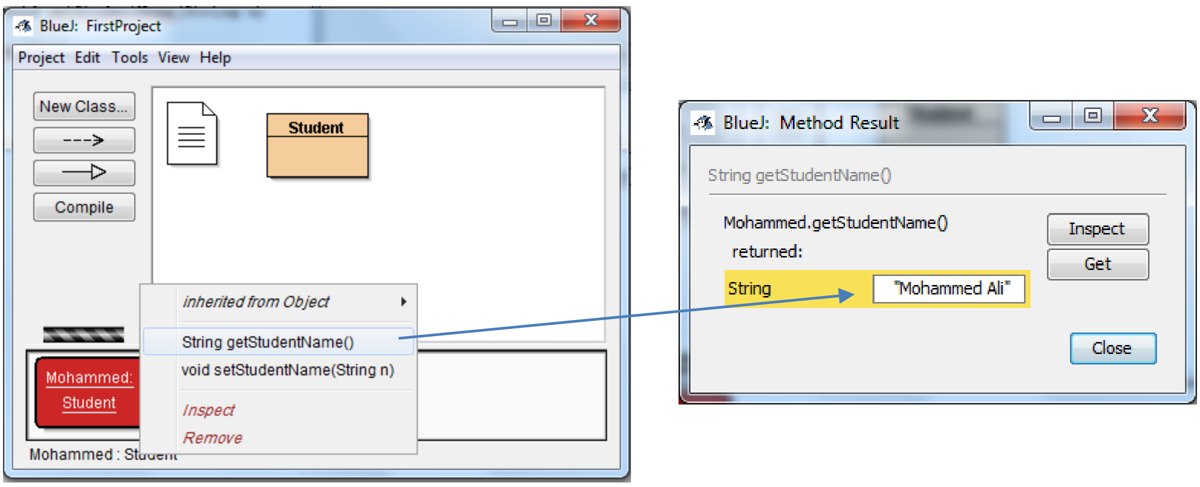


هذا الاسم *"Mohammed Ali"* الذي ادخلناه سيتم تخزينه في المتغير n كالتالي :

```
public void setStudentName(String n)
{
    studentName = n ;
}
```

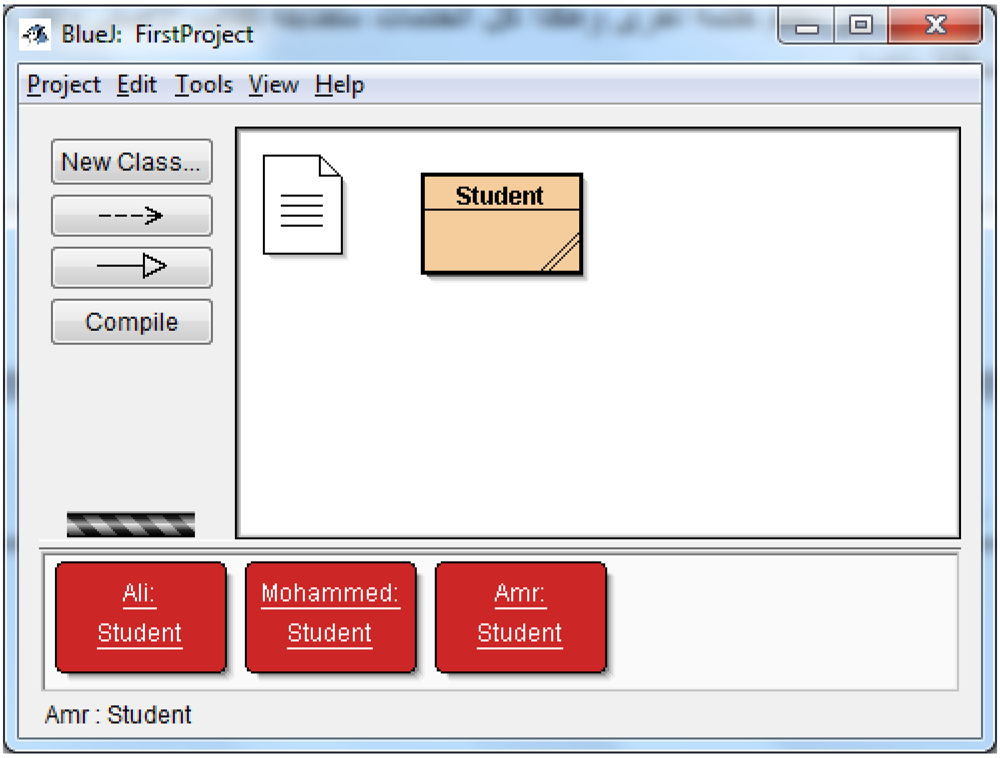


ومن ثم سيخزن في الحقل *studentName* . وبهذا اذا ما استرجعنا هذا الحقل باستخدام دالة *getStudentName()* سوف ترجع لنا هذا القيمة كالتالي :



لاحظ انه يمكننا ان نكون اكثر من كائن باستخدام الفصيل *Student* فمثلا يمكننا ان نكون مثيل اسمه *"Ali"* وكائن اخر اسمه *"Mohammed"* وثالث اسمه *"Amr"* ... وهكذا . كل كائن له حقوله ومناهجه الخاصة وليس له علاقة بالكائن الاخر . هذ الامر يشبه تماما وكأننا صنعنا ختم (وهو الفصيل **Student** هنا) وفي كل كائن يتم تكوينه كأننا نختم ختمه بالختم على الورق فاذا اردنا كائن اخر نختم ختمة اخرى وهكذا كل الختمات متطابقة للقالب الاصلي ولكن كل ختمة مستقلة بذاتها.

بعد تكوين عدة كائنات ستظهر منصة العمل كالتالي :



لاحظ تكوين عدة كائنات بنفس الفصيل وكل كائن مستقل عن الاخر بحقوله ومناهجه.

المثال الثانى :

عدل البرنامج السابق واجعله يحتوي على حقلين نوعهما خاص هما حقل اسم الطالب *(studentName)* والدرجة *(mark)* على ان يتم تطبيق مفاهيم البرمجة الهدفية *(OOP)* .

سنقوم بإضافة حقل جديد اسمه ***mark*** بعد الحقل السابق نوعه كسري *double* وامكانية الوصول هو خاص *private* .

```
private double mark;
```

ايضا سنقوم بتصفير هذا الحقل ضمن دالة الباني كالتالي :

```
mark = 0;
```

لتكون البرمجة كالتالي :

```
public class Student
{

    private String studentName;
    private double mark;

    public Student()
    {
         studentName = "";
         mark = 0;
    }

    public String getStudentName()
    {
       return (studentName);
    }

    public void setStudentName(String n)
    {
       studentName = n ;
    }
}
```

ايضا سنضيف دالتين احداها هي لترجيع قيمة الدرجة وليكن اسمها *getMark()* ستكون عامة *public* وترجع بقيمة كسرية *double* ، والدالة الاخرى هي لتخزين قيمة في حقل الدرجة وليكن اسمها *setMark(double m)* حيث انها سوف لن ترجع أي قيمة *void* وسوف تستقبل قيمة كسرية في الباراميتر *m* ليتم تخزينها فيما بعد في حقل الدرجة *mark* كالتالي :

برمجة دالة الـ(get)

```
    public double getMark()
    {
       return (mark);
    }
```

برمجة دالة الـ(set)

```
    public void setMark(double m)
    {
       mark = m ;
    }
```

لتكون البرمجة كاملة كالتالي :

```
public class Student
{
    private String studentName;
    private double mark;

    public Student()
    {
        studentName = "";
        mark = 0;
    }

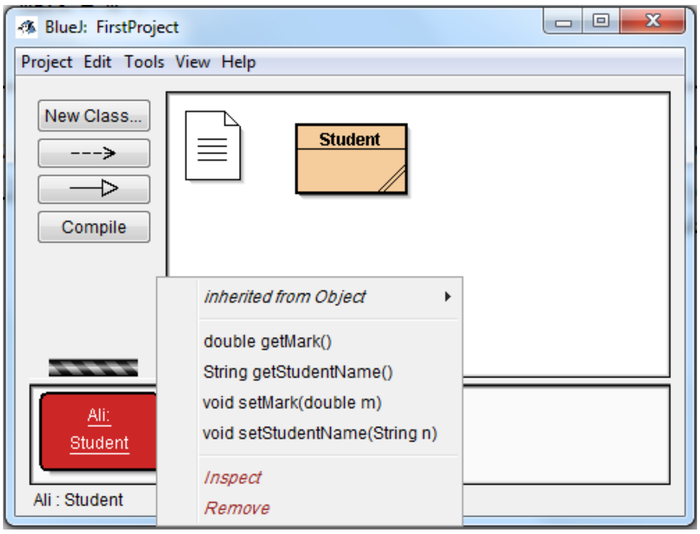
    public String getStudentName()
    {
       return (studentName);
    }

    public void setStudentName(String n)
    {
       studentName = n ;
    }

    public double getMark()
    {
       return (mark);
    }

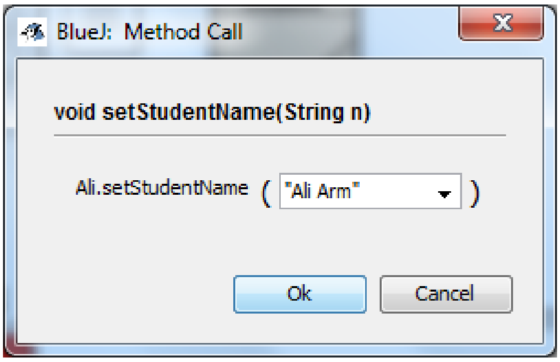
    public void setMark(double m)
    {
       mark = m ;
    }
}
```

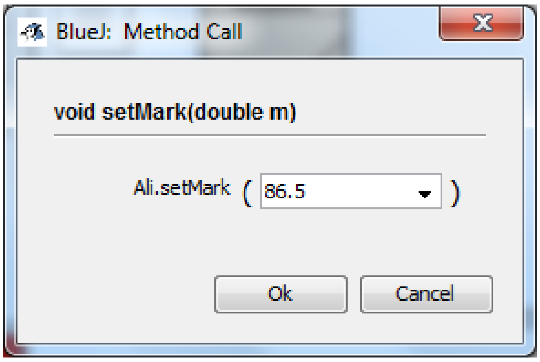
الان يمكننا تكوين كائنات (objects) جديدة واعطاءها قيم لكل من حقلي الاسم والدرجة بواسطة دوال الـ(set) ومن ثم استرجاعها بواسطة دوال الـ(get) ولا يمكن الوصول الى الحقول مباشرة لأنها خاصة.



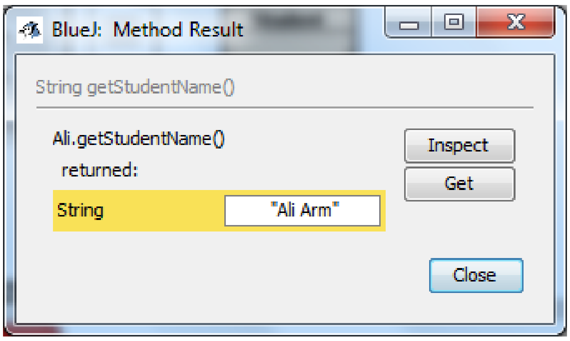
لاحظ ظهور اربعة مناهج التي قمنا ببرمجتها ، وظهرت بالقائمة لأنها عامة *public* فلو غيرت احدها الى خاصة *private* سوف لن تظهر بالقائمة بعد ذلك.

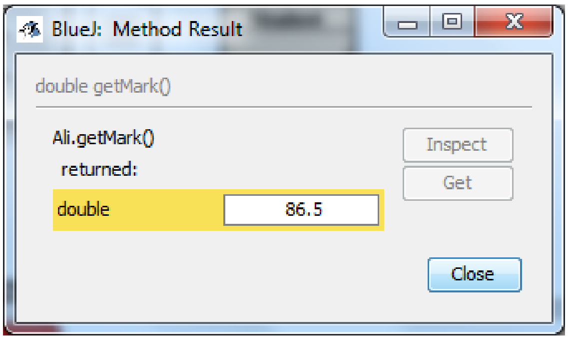
فلو كونا كائن جديد *object* اسمه Ali واخترنا من القائمة دالة تعديل الاسم *setStudentName(String n)* وكذلك دالة تعديل الدرجة *setMark(double m)* فسوف يظهر لنا مربعي حوار لكل منهما يطلب منك سلسلة للاسم *n* ورقم كسري يمثل الدرجة *m* والتي سيتم تخزينها في حقلي الاسم *studentName* والدرجة *mark* كالتالي :



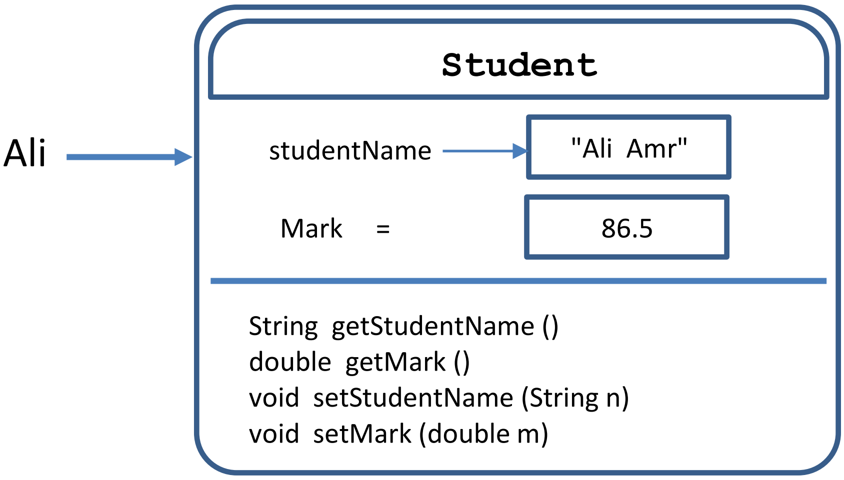


الان اذا اخترنا دالتي ترجيع الاسم *getStudentName()* وكذلك الدرجة *getMark()* من القائمة فسيعطيك اخر اسم ودرجة قمت بإدخالها (وهي "Ali Arm" و 86.5) كالتالي:





أي ان الكائن Ali هو من نوع الـStudent والذي يحتوي بداخله على حقلين هما اسم الطالب *studentName* ودرجته *mark* وكذلك عدة مناهج او دوال كما موضح بالشكل :

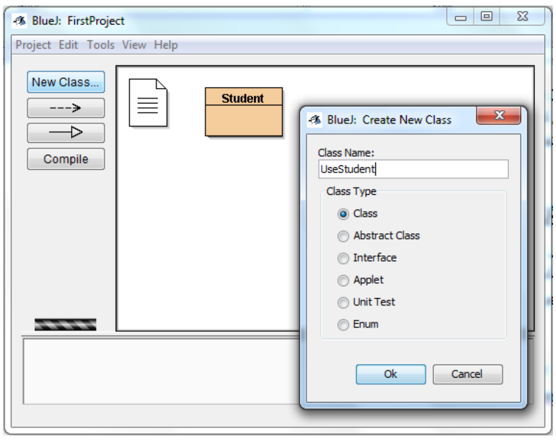


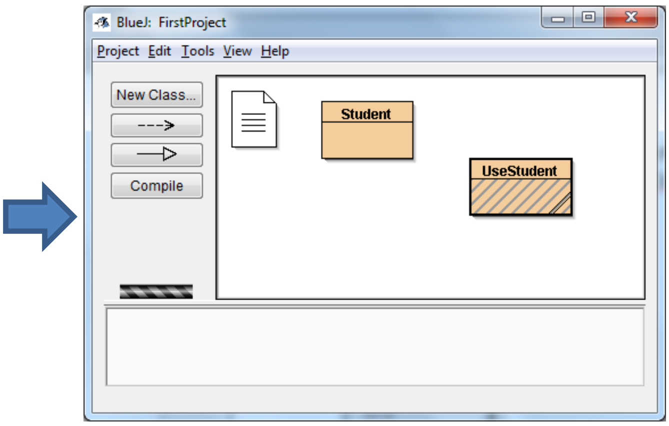
الان بعد ان قمنا بتكوين الكائنات *objects* باستخدام الـ *blueJ* بشكل مبسط سوف ننتقل فيما يلي الى كيفية تكوين كائنات باستخدام البرمجة وكذلك كيفية بناء اكثر من فصيل واحد وكذلك كيفية استخدام فصيل لكائنات فصيل اخر عن طريق الوصول او استدعاء مناهجه العامة .

المثال الثالث :

كون فصيل جديد اسمه (**UseStudent**) بواسطته يتم تكوين كائنات (*objects*) من فصيل الطالب (*Student*) الذي كونته قبل قليل . ومن ثم استدعاء مناهجه او دواله (دوال الـ set) لتغيير اسم الطالب ودرجته ، وطباعة حقوله عن طريق استدعاء دوال الـ(get).

- الان سنقوم فصيل جديد وذلك بالنقر على زر (*New Class ...*) واعطاءه اسمه (*UseStudent*) كالتالي :





- انقر نقرا مزدوجا على الفصيل الجديد واجعل البرمجة به كما يلي :

```
public class UseStudent
{

    public UseStudent()
    {


    }

}
```

➢ ضمن باني فصيل *UseStudent* كون برمجيا كائن جديد *Object* (أي طالب وليكن **ali**) من فصيل الطالب *Student* كالتالي :

```
Class_name    object = new Class_name();
```

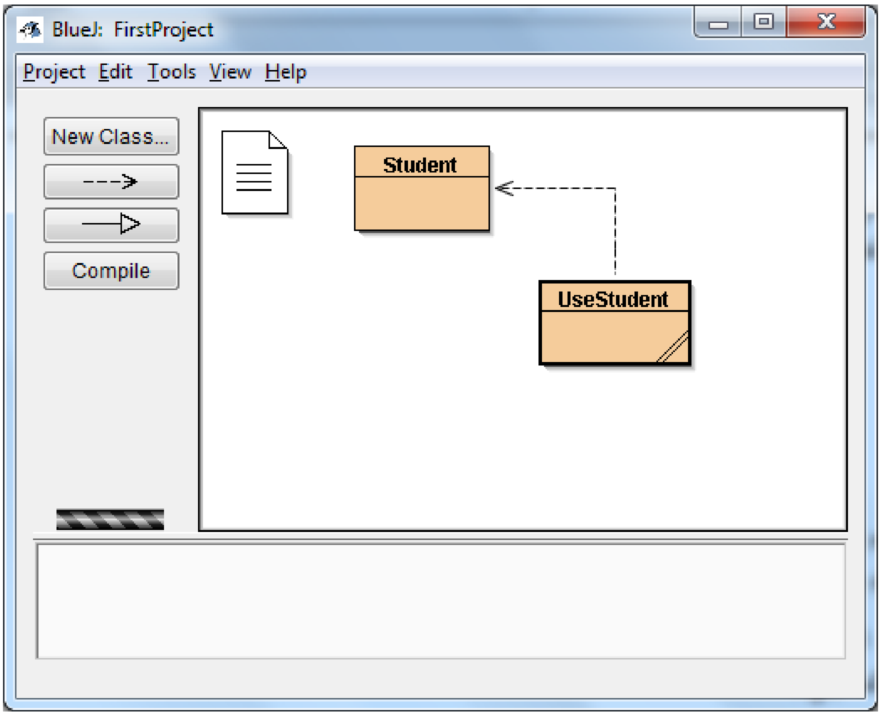
أي :

```
Student ali = new Student();
```

أي نكتب اسم الفصيل اولا ثم يتبعه اسم للكائن الجديد وليكن (*ali* وهو يخضع لشروط المعرفات في الجافا) ثم علامة (=) ثم كلمة جديد (new) ثم استدعاء لباني فصيل الطالب متبوعا بقوسين .

```
public class UseStudent
{
    public UseStudent()
    {
        Student ali = new Student();
    }
}
```

اذا ترجمت البرنامج الان سوف تلاحظ سهم يخرج من فصيل *UseStudent* الى فصيل *Student* كما يلي :



لوضع قيم لاسم الطالب ودرجته نكتب ما يلي :

```
Object_name . method_name(arguments);
```

أي نكتب :

```
ali.setStudentName("Ali Ahmed");
```

أي نكتب اسم الكائن (هنا *ali*) متبوعا بنقطة ثم اسم المنهج (هنا *setStudentName ()*) ، يليه قوسين تحتوي على القيم المراد تمريرها الى المنهج (هنا الاسم *"Ali Ahmed"*) .

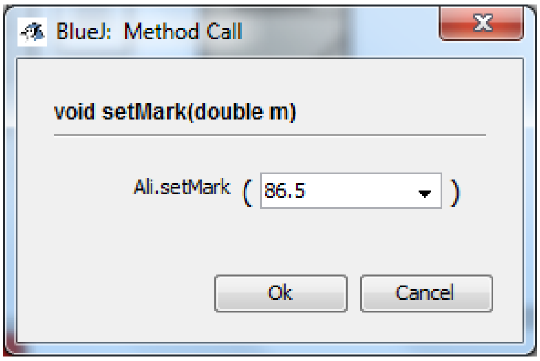
وكذلك الحال بالنسبة للدرجة نكتب التالي :

```
ali.setMark(94.2);
```

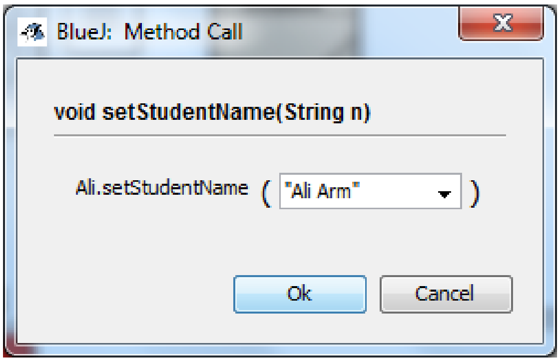
لاحظ ان السلسلة يجب حصرها بين علامتي التنصيص المزدوجة وان الارقام تكتب بدون علامتي تنصيص.

و هي تقابل تماما ما فعلناه في السابق من ادخال قيم عن طريق الـbluej كالتالي :

```
ali.setMark(86.5);
```




```
ali.setStudentName("Ali Amr");
```



اما لاسترجاع قيم الاسم والدرجة فإننا نكتب اسم الكائن متبوعا بنقطة ثم استدعاء لمنهج الحصول على الاسم (*getStudentName()*) حيث يمكن كتابتها ضمن جملة الطباعة مباشرة كالتالي :

```
System.out.println( ali.getStudentName() );
```

وكذلك الحال بالنسبة للحصول على درجة الطالب :

```
System.out.println( ali.getMark() );
```

عليك دائما استخدام ايعاز تنظيف الشاشة عند الحاجة وقبل البدء بالطباعة وهو :

```
System.out.println( '\u000C' );
```

لتصبح البرمجة كاملة كالتالي :

```
public class UseStudent
{

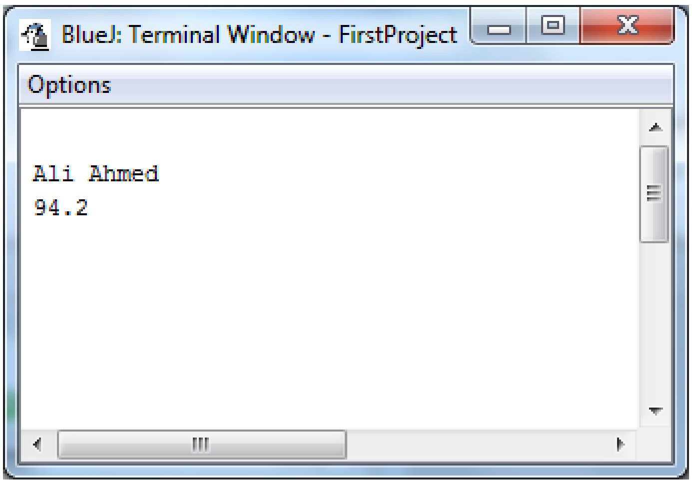
    public UseStudent()
    {

          Student ali = new Student();

          ali.setStudentName("Ali Ahmed");
          ali.setMark(94.2);

          System.out.println( '\u000C' );
          System.out.println( ali.getStudentName() );
          System.out.println( ali.getMark() );
    }
}
```

عند تنفيذ البرنامج وتكوين كائن جديد من الفصيل *UseStudent* تكون النتيجة كالتالي :

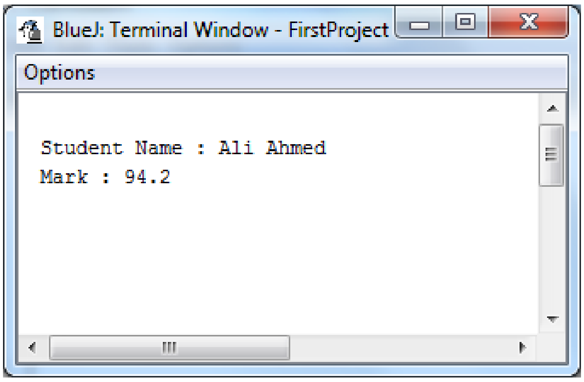


لاحظ أن السلاسل المطبوعة كانت مبهمة المعنى ما لم يتم ايضاحها بسلسلة قبلها تبين معنى كل قيمة كالتالي :

```
System.out.println(" Student Name : " +  ali.getStudentName() );

System.out.println(" Mark : " + ali.getMark() );
```

لتكون النتيجة :



لاحظ اننا استخدمنا باني الفصيل *UseStudent* لكتابة الجمل البرمجية السابقة ولكن عندما يكون المشروع مكون من عدة فصائل عادة ما يكون هناك فصيل يقود هذه الفصائل ، فلكون مشروعنا يحتوي على فصليين فقط هم *Student* والفصيل الذي يستخدم هذا الفصيل وهو *UseStudent* فان الفصيل الذي سيقود بقية الفصائل او يختبرها يجب ان يحتوي على المنهج الرئيسي **main()** وبذلك يمكن الاستغناء عن الباني في فصيل *UseStudent* ، ويكون بالصيغة التالية :

```
public static void main(String[] args)
{

}
```

اسم الدالة هي main وهي تقوم بنفس عمل دالة الـmain في لغة السي ، هذه الدالة تكون عامة public حتى يمكن الوصول اليها من خارج الفصيل ، وان هذه الدالة سوف لن ترجع قيمة void ، المعاملات (String[] args) يكون مفيدة لتمرير قيمة للدالة كما سيوضح لاحقا.

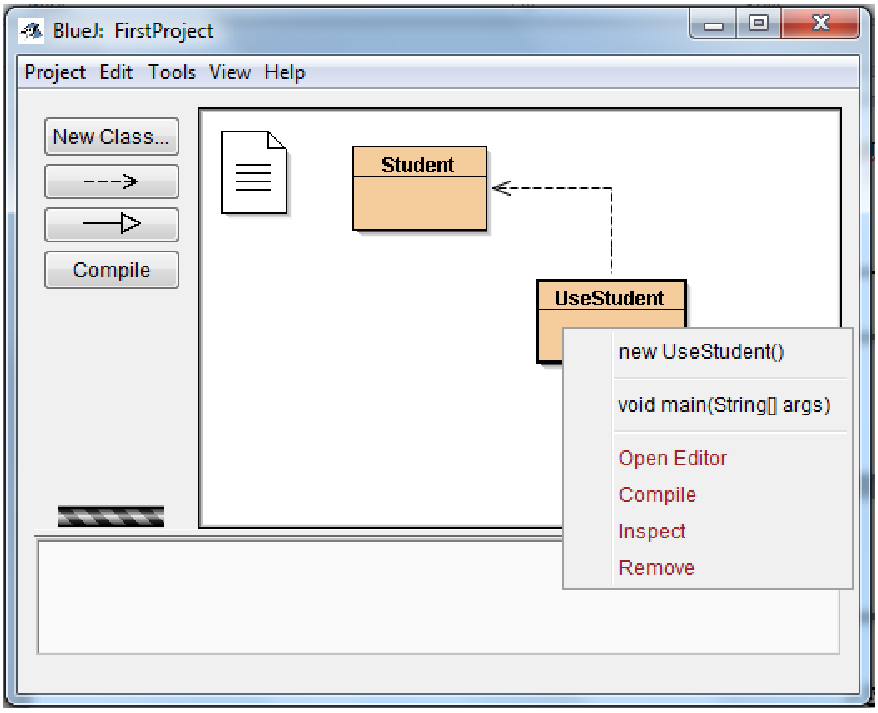
لاحظ استخدام كلمة static والذي يعني امكانية استخدام هذا المنهج مع اسم الفصيل ودون الحاجة الى تكوين كائن منه.

مثل كل منهج تبدأ الدالة بقوس فئة مفتوح { وينتهي بقوس فئة مغلق } .

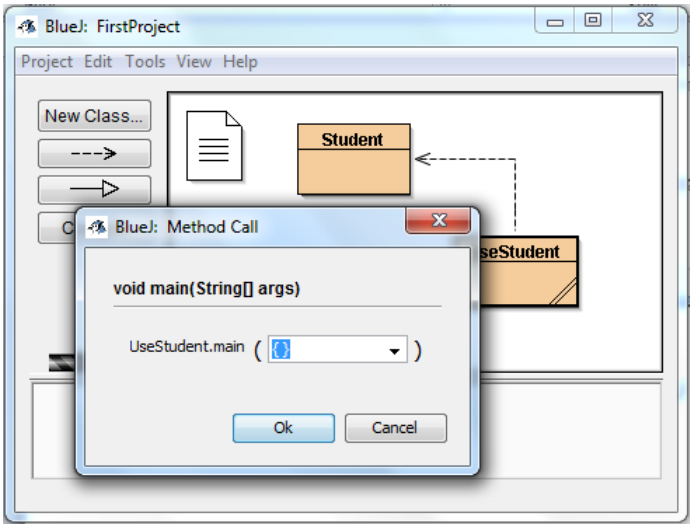
```
public class UseStudent
{
    public static void main(String[] args)
    {
        Student ali = new Student();
        ali.setStudentName("Ali Ahmed");
        ali.setMark(94.2);

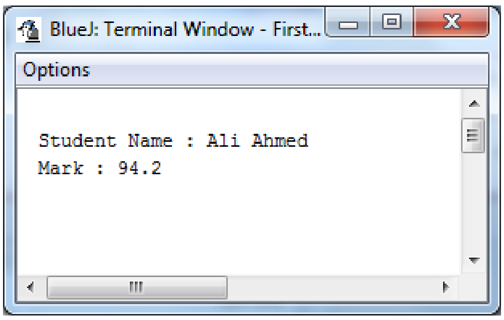
        System.out.println( '\u000C' );
        System.out.println( ali.getStudentName() );
        System.out.println( ali.getMark() );
    }
}
```

لاحظ أن عند ترجمة البرنامج والنقر بالزر الايمن على الفصيل UseStudent يظهر المنهج main كالتالي :



وعند اختيار المنهج main سيظهر مربع حوار لادخال معاملات للـargs انقر فقط على ok وبدون ادخال أي شيء ، كما تلاحظ ان الدالة تم تنفيذها مباشرة وبدون تكوين كائن في منصة العمل كالتالي :





حل التدريب (1)

اسم المعرف	هل هو صحيح أو لا	في حالة (لا) اذكر السبب
Private	صحيح	
$name	صحيح	
_Quantity	صحيح	
five_5	صحيح	
Windows 7	خطأ	لأنه يحتوي على فراغ
c#	خطأ	لأنه يحتوي على رمز خاص
sql	صحيح	
$_88	صحيح	
item-8	خطأ	لأنه يحتوي على رمز خاص (-)

حل التدريب (2)

`System.out.println ( 120 + "Hello");`	120Hello
`System.out.println ( 120 + 40 + "Hello");`	160Hello
`System.out.println ( 120+"Hello" + 60);`	120Hello60
`System.out.println ( "Hello" + 120 + 40);`	Hello12040
`System.out.println ( "Hello" + (120 + 40) );`	Hello160

التحميل الزائد للمنهج Overloading :

واحد من اهم مفاهيم البرمجة الكائنية OOP هي ما يسمى بالتحميل الزائد للمنهج overloading وهو ما يعني ان هناك اكثر من منهج او دالة داخل الفصيل الواحد **لها نفس الاسم مع اختلاف في عدد ونوع البارامترات** ولتوضيح هذا المفهوم سنقوم بتعديل البرنامج السابق :

فمثلا ضمن فصيل الطالب السابق كان هناك باني واحد ولا يستقبل معاملات وهو كالتالي :

```
        ................................

private double mark;

public Student()
{
     studentName = "";
     mark = 0;
}

public String getStudentName()
{
    ................................
```

يمكننا تكوين باني اخر بنفس الاسم والذي يستقبل فقط اسمة الطالب اضافة الى الباني الاول كالتالي :

```
        ................................

private double mark;

public Student()
{
     studentName = "";
     mark = 0;
}

public Student(String n)
{
     studentName = n;
     mark = 0;
}

public String getStudentName()
{
          ................................
```

لاحظ ان المنهج الثاني (الباني الثاني) يستقبل معامل من نوع السلسلة الرمزية اسمه n ويضعه في الحقل *studentName* بينما الدرجة *mark* بقت كما كانت ووضعنا بها صفر.

ايضا يمكن وضع باني ثالث بنفس الاسم مع معاملين هما للاسم والدرجة كالتالي :

```
public class Student
{
    private String studentName;
    private double mark;

    public Student()
    {
        studentName = "";
        mark = 0;
    }

    public Student(String n)
    {
        studentName = n;
        mark = 0;
    }

    public Student(String n, double m)
    {
        studentName = n;
        mark = m;
    }

    public String getStudentName()
    {
       return (studentName);
    }

    public void setStudentName(String n)
    {
       studentName = n ;
    }

    public double getMark()
    {
       return (mark);
    }

    public void setMark(double m)
    {
       mark = m ;
    }
}
```

لاحظ ان جميع الدوال لها نفس الاسم مع اختلاف عدد ونوع المعاملات فيمكننا استدعاء أي منهج من الثلاثة السابقة عن بناء الكائن فمثلا في بداية العام الدراسي حيث انه لا يمكننا معرفة اسماء الطلبة الجدد بعد ولا درجاتهم ، في هذه الحالة يمكننا استدعاء الباني الاول بدون معاملات كالتالي :

```
Student ali = new Student();
```

- ثم بعد ذلك عندما يقوم الطالب بالتسجيل يتم معرفة اسمه فيمكن تعديل اسمه عن طريق استدعاء المنهج *setStudentName()* كالتالي :

```
ali.setStudentName("Ali Ahmed");
```

بعدها عندما يقوم بإجراء امتحان ورصد درجة له يمكننا تعديل الدرجة باستدعاء المنهج *setMark()* كالتالي :

```
ali.setMark(80.2);
```

- أما اذا كونا الكائن بعد تسجيل الطالب وقبل القيام بالاختبار فيمكننا تكوين كائن مع استدعاء الباني الثاني كالتالي :

```
Student ali = new Student("Ali Ahmed");
```

هنا عينا اسم للطالب ورصدت 0 للدرجة في بداية الامر ، ثم عند قيام الطالب باختبار يمكن تعديل الدرجة له كالتالي :

```
ali.setMark(80.2);
```

- اما اذا كونا الكائن (*object*) في نهاية العام الدراسي أي بعد تسجيل الطالب ومعرفة اسمه وايضا بعد قيامه بالامتحانات ومعرفة نتائجه فيمكننا تكوين كائن باستدعاء الباني الثالث كالتالي :

```
Student ali = new Student("Ali Ahmed", 80.2);
```

لاحظنا من المثال السابق استخدامنا لنفس الاسم مع اختلاف عدد المعاملات :

`Student ali = new Student();`	**استدعاء منهج الباني بدون معاملات**
`Student ali = new Student("Ali Ahmed");`	**استدعاء منهج الباني بمعامل واحد**
`Student ali = new Student("Ali Ahmed", 80.2);`	**استدعاء منهج الباني بمعاملين**

مثال اخر :

هذا البرنامج سيقوم بتكوين فصيل اسمه الجمع Addition يقوم بتنفيذ عملية الجمع .

- اولا : سنقوم بتكوين فصيل جديد اسمه Addition مع ترك الباني كالتالي :

```
public class Addition
{
    public Addition()
    {

    }
}
```

- الان سنقوم بتكوين دالة عامة (public) اسمها sum والتي تأخذ معاملين صحيحين (a, b) وترجع قيمة صحيحة (int) هي حاصل جمعهما القيمتين المرسلتين كالتالي :

```
    public int sum(int a, int b)
    {
        return (a + b);
    }
```

```
public class Addition
{
    public Addition()
    {

    }

    public int sum(int a, int b)
    {
       return (a + b);
    }
}
```

- الان سنقوم بتكوين دالة اخرى لها نفس الاسم (sum) وتستقبل ثلاثة معاملات صحيحة (a, b, c) وترجع حاصل جمعها كالتالي :

```
public class Addition
{
    public Addition()
    {

    }

    public int sum(int a, int b)
    {
       return (a + b);
    }

    public int sum(int a, int b, int c)
    {
       return (a + b + c);
    }
 }
```

➢ الان سنقوم بتكوين دالة اخرى لها نفس الاسم (sum) وتستقبل اربعة معاملات حقيقية (a, b, c, d) وترجع قيمة حقيقية هي حاصل جمع الاربعة اعداد كالتالي :

```
public class Addition
{
    public Addition()
    {

    }

    public int sum(int a, int b)
    {
       return (a + b);
    }

    public int sum(int a, int b, int c)
    {
       return (a + b + c);
    }

    public double sum(double a, double b, double c, double d)
    {
       return (a + b + c + d);
    }

 }
```

➢ الان سنقوم بتكوين فصيل جديد اسمه *UseAddtition* مع تكوين دالة *الـmain* كالتالي :

```
public class UseAddition
{
    public static void main(String[] args)
    {

    }
}
```

ضمن دالة الـmain سنقوم بتكوين كائن جديد من الـAddition اسم op كالتالي :

```
Addition op = new Addition();
```

الان يمكننا استدعاء أي من دوال الجمع (*sum*) كالتالي :

```
op.sum(20, 30);
```

او يمكننا ارسال 3 معاملات صحيحة كالتالي :

```
op.sum(20, 30, 60);
```

او يمكننا ارسال اربعة قيم كسرية كالتالي :

```
op.sum(5.6, 2.4, 8.1, 4.4);
```

فالمبرمج يشعر وكأنه يستدعي نفس الدالة ولكن بمعاملات مختلفة في النوع والعدد ، فالبرنامج بعد تكملته *كالتالي :*

```
public class UseAddition
{
  public static void main(String[] args)
  {
    Addition op = new Addition();

    System.out.print('\u000C');

    System.out.println("Sum 1 : " + op.sum(20, 30) );
    System.out.println("Sum 2 : " + op.sum(20, 30, 60) );
    System.out.println("Sum 3 : " + op.sum(5.6, 2.4, 8.1, 4.4) );
  }
}
```

لاحظ اننا عند استدعاء أي منهج (دالة) من فصيل اخر نكتب اسم الكائن من الفصيل الاخر (op) أولاً ثم نقطة ثم اسم المنهج (sum) ثم نمرر قيم للمعاملات بين القوسين :

op . sum (20, 30)

لاحظ انه عندما استدعينا دالة sum بإرسال قيمتين صحيحتين سيتم استقبالها ضمن الدالة التي لها معاملين صحيحين كالتالي :

```
        op.sum(20, 30);            نقطة الاستدعاء

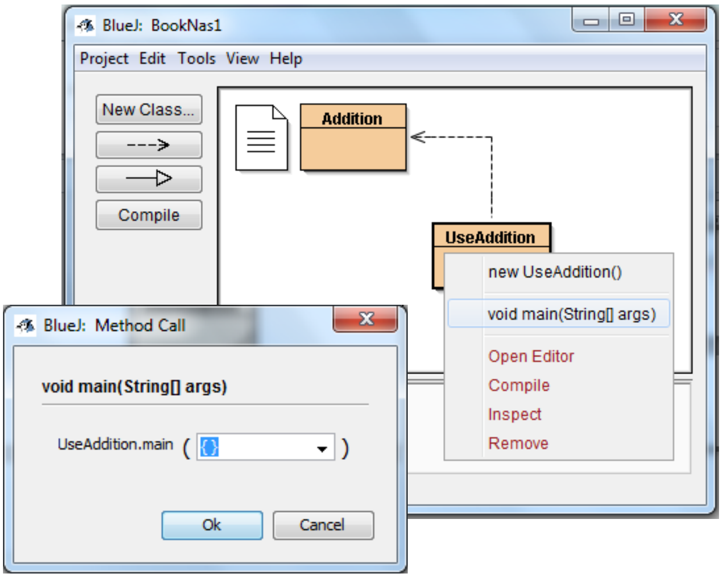
public int sum(int a, int b)       الاستقبال
{
    return (a + b);
}
```

وعندا استدعائنا لدالة *sum* من الكائن *op* مع ارسال اربعة قيم حقيقية سيتم استقبال القيم عن طريق دالة الـ *sum* والتي تحتوي على اربعة معاملات حقيقية كالتالي :

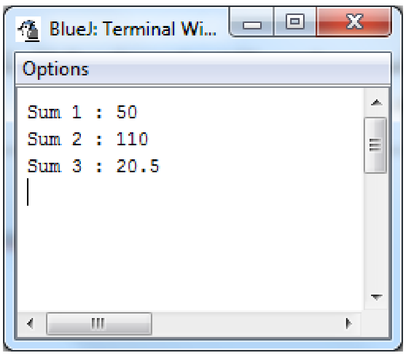
```
        op.sum(5.6,   2.4,   8.1,   4.4);

public double sum(double a, double b, double c, double d)
{
    return (a + b + c + d);
}
```

الان عند تنفيذ البرنامج واختيار الـmain يظهر مربع الحوار التالي ، انقر على ok كالتالي :



فتكون النتيجة كالتالي :



لاحظ انه عندما كونا منهج او دالة جديدة كتبنا **نوع** الدالة هل هي (عامة او خاصة او ...الخ) ثم **القيمة التي سترجعها** الدالة (سلسلة او قيمة صحيحة او حقيقية او ... الخ) ثم **اسم الدالة** في مثالنا السابق كان *sum* وهو يخضع لشروط تكوين المعرفات في الجافا ، ثم قوسين بينها **نوع واسماء البارامترات** التي سيتم استقبالها مفصولة عن بعضها بفواصل ، ثم تحصر جمل الدالة بين قوسي الفئة { } كالتالي :

نوع القيمة المرجعة

نوع البارامير

نوع الدالة

اسم الدالة

```
public   double   sum   (int a,   int b)
{
        جملة 1 ;
        جملة 1 ;
        جملة 1 ;
        return   قيمة ;
}
```

اسم البارامير

تكتب دائما اذا كانت الدالة ترجع قيمة ، اما اذا كانت لا ترجع قيمة فتكتب void.

يمكن ان تحتوي الدالة الواحدة على اكثر من **return** واحدة ، فعند الوصول اليها يتم ترجيع القيمة والخروج من الدالة .

__تجاوز المناهج Overriding Methods :__

تجاوز المناهج يعتبر ايضا واحد من مفاهيم البرمجة الكائنية OOP ، حيث اعادة برمجة مناهج ثم تكوينها ضمن الفصيل الاب ضمن احد الفصائل الابناء ضمن ما يعرف بالوراثة ، سيتم توضيح ذلك اكثر عند شرح موضوع الوراثة Inheritance ، الان لفهم الـ*overriding* لنفرض انه ضمن مثال الطالب السابق كونا كائنين من فصيل الطالب كالتالي :

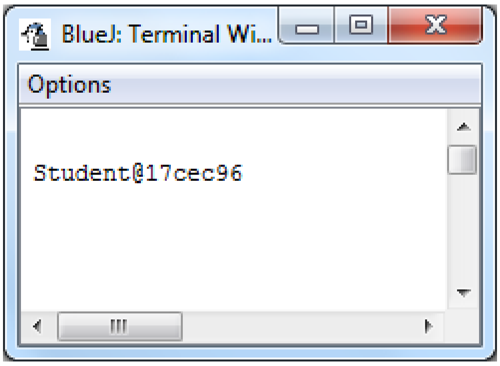
```
Student std1 = new Student("Ali Ahmed", 80);

Student std2 = new Student("Ali Ahmed", 80);
```

ماذا سيكون ناتج جملة الطباعة التالية :

```
System.out.println(std1);
```

النتيجة ستكون كالتالي :



و هو كلام غير مفهوم ، وكنا نتوقع أن يتم طباعة اسم الطالب ودرجته بتنسيق معين !!

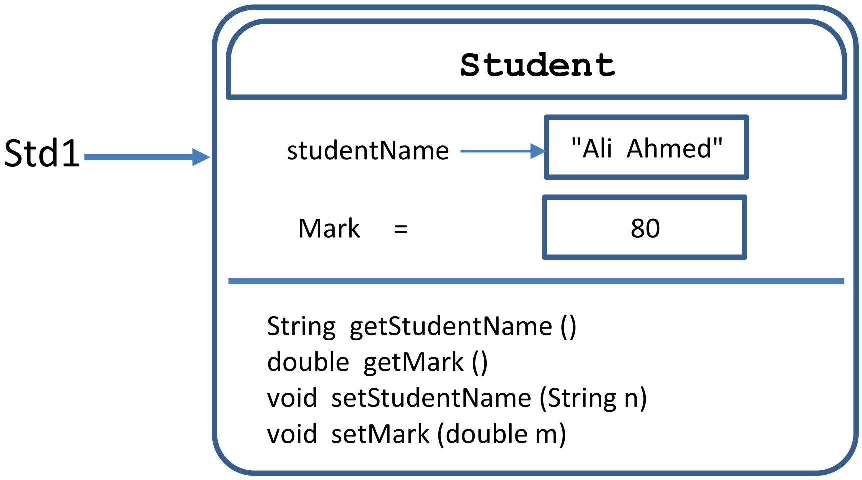
أيضا ، هل std1 == std2 ؟؟؟ بمعنى ماذا ستكون النتيجة ضمن جملة الطباعة التالية :

```
System.out.println(std1 == std2);
```

النتيجة هي خطأ ***False*** ولكن لماذا رغم ان القيم المرسلة ضمن الكائن الاول ***std1*** هي مساوية للقيم المرسلة ضمن الكائن الثاني ***std2*** ؟؟

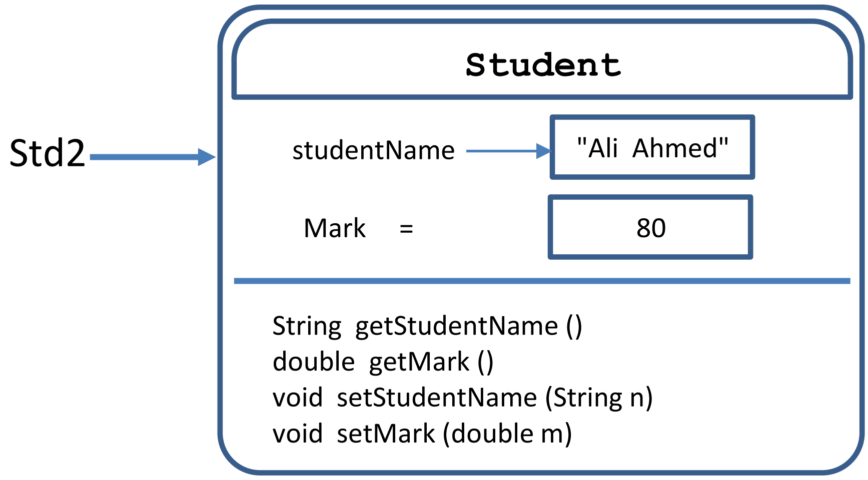
في جملة تكوين الكائن الاول الـ**std1** سيؤشر الى كائن كالتالي:

```
Student std1 = new Student("Ali Ahmed", 80);
```



أما الجملة الثانية فهي ستؤشر الى كائن اخر منفصل عن الاول ولا علاقة له به كالتالي:

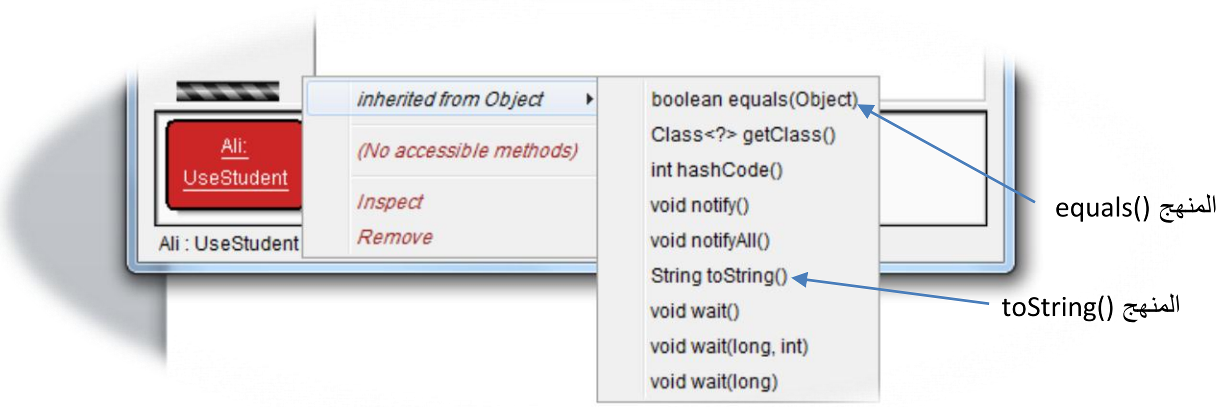
```
Student std2 = new Student("Ali Ahmed", 80);
```



أي ان كل كائن سيكون مرجع مفصول عن الاخر.

ولكن منطقيا، كيف نجعل std1==std2 ؟ لكونهما يحتويان على نفس القيم بغض النظر الى انهما يؤشران الى كائنين مختلفين. ايضا كيف نجعل القيم الناتجة من طباعة الكائن منطقية ومفهومه لتمثل اسم ودرجة الطالب بدلا من الكلام الغير واضح عند طباعة الكائن ؟

كل الفصائل التي يتم تكوينها في الجافا تكون موروثة من فصيل يسمى **Object** بما فيها الفصائل التي نكونها بأنفسنا ، لذلك ومجانا سترث العديد من المناهج والتي تم اعدادها مسبقا ضمن الكائن ***Object*** حيث ان هناك العديد من المناهج التي نرثها من هذا الكائن ونستطيع اعادة برمجتها وفق ما نريد ، يمكنك مشاهدة هذه المناهج عند النقر بالزر الايمن على الكائن المكون ضمن منصة العمل في الـ blueJ لتظهر بند في بداية القائمة يوضح المناهج التي تم وراثتها من الكائن Object (*Inherited from Object*) كالتالي :



الان لنعيد برمجة المنهج **toString()** والموروث من فصيل الـ Object ضمن فصيل الطالب Student كالتالي :

```
public String toString()
{
      return (" Name : " + studentName + "\n Mark : " + mark);
}
```

لاحظ ان اعادة برمجة المنهج ضمن الفصيل الابن (*Student*) تكون بنفس اسم المنهج الموجود بالفصيل الاب (*Object*) وبنفس نوع واسم المعاملات الموجودة بالفصيل الاب وهذا ما يسمى بتجاوز المناهج (*Overriding Methods*).

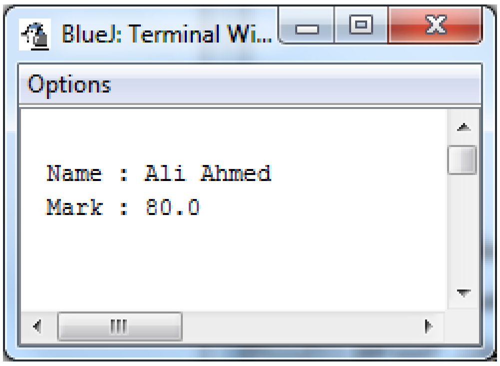
الآن عند طباعة الكائن *std1* ضمن جملة الطباعة في الفصيل *UseStudent* كالتالي :

```
System.out.println(std1);
```

هي ضمنيا استدعاء للمنهج *toString()* أي يمكن كتابتها كالتالي والنتيجة ستكون واحدة :

```
System.out.println( std1.toString() );
```

والنتيجة ستكون مفهومة وكما قمنا بتنسيقها ضمن المنهج *toString()* كالتالي :



ملاحظة : الرمز **"\n"** هو احد رموز الهروب للانتقال الى سطر جديد.

ايضا يمكننا جعل (std1==std2) لكون تحمل نفس القيم وذلك بتجاوز منهج **equals()** الموجود ضمن الكائن الاب *Object* واعادة برمجته ضمن المنهج الابن *Student* كالتالي :

```
public boolean equals(Student s)
{
      return ( (studentName == s.studentName)
                 && (mark == s.mark) );
}
```

لاحظ ان الدالة ترجع قيمة منطقية *boolean* وهي (***true*** او ***false***) ، هنا قمنا بمقارنة حقل اسم الطالب *studentName* مع حقل اسم الطالب والذي جاء ضمن الكائن ***S*** كباراميتر (***s.studentName***) فاذا كانتا متساويتان وكذلك الحال للدرجتين فان النتيجة ستكون صحيحة (*true*) والا فالدالة سترجع القيمة (*false*).

لكون اسم الطالب (*studentName*) من نوع فصيل السلسلة (*String*) فيمكننا استخدام المنهج equals() الخاصة بفصيل السلاسل لمقارنة سلسلتين كالتالي :

```
studentName.equals(s.studentName)
```

بدلا من :

```
studentName == s.studentName
```

هناك مغالطة ضمن ما سبق برمجته وهو ان منهج equals() ضمن الفصيل الاب (Object) هي كالتالي :

```
public boolean equals(Object obj)
{
}
```

ونحن استخدمنا :

```
public boolean equals(Student s)
{
}
```

ولكنها ستقوم بالمطلوب ، فقط لوكوننا لم نصل الى شرح التحويل القصري بين الكائنات بعد أي بين Object والـ Student ، ولكنها ستعمل وبدون مشاكل.

لاحظ ايضا اننا استخدمنا حقول الكائن (s) وهي اسم الطالب (*s.studentName*) ودرجته (*s.mark*) رغم انها حقول خاصة (***private***) !! وهذا مسموح به لكوننا نعمل داخل نفس الفصيل Student .

فصيل الطالب Student كاملا هو :

```
public class Student
{
    private String studentName;
    private double mark;
    public Student()
    {
        studentName = "";
        mark = 0;
    }

    public Student(String n, double m)
    {
        studentName = n;
        mark = m;
    }

    public String getStudentName()
    {
       return (studentName);
    }

    public void setStudentName(String n)
    {
       studentName = n ;
    }

    public double getMark()
    {
       return (mark);
    }

    public void setMark(double m)
    {
       mark = m ;
    }

    public String toString()
    {
         return (" Name : " + studentName +
                 "\n Mark : " + mark);
    }

    public boolean equals(Student s)
    {
         return ( (studentName == s.studentName)
                   && (mark == s.mark) );
    }

}
```

اما الفصيل (UseStudent) فهو :

```
public class UseStudent
{
    public static void main(String[] args)
    {

        System.out.println( '\u000C' );

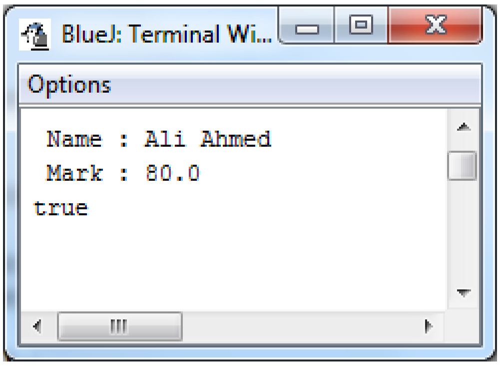
        Student std1 = new Student("Ali Ahmed", 80);
        Student std2 = new Student("Ali Ahmed", 80);

        System.out.println( std1 );

        System.out.println( std1.equals(std2) );

    }
}
```

و النتيجة ستكون :



التغليف أو الكبسلة Encapsulation :

واحد من مبادئ البرمجة الشيئية OOP هي الكبسلة ، والتي تعني ان الفصيل يكون مغلفا محتويا على حقول ودوال لا يستطاع الوصول الى حقوله وتغييرها مباشرة الا عن طريق استدعاء المناهج المخصصة لذلك ، بغض النظر عن تفاصيل البرمجة ضمن الفصيل ، فقط كل ما يهمنا هو استدعاء المناهج المطلوب لتنفيذ وظيفة معينة.

لتحقيق مبدا الكبسلة للفصائل لابد من وضع محددات (Modifiers) للتحكم في الوصول الى حقول ومناهج الفصيل و هي اربعة محددات كالتالي :

1- عام public.

2- خاص private .

3- محمي protected .

4- افتراضي default .

هذه المحددات الاربعة يمكن وضعها مع **اسماء الحقول** او **المناهج** لتحديد درجة الوصول اليها .

1- المحدد العام public :

يعني ان المتغير *Variable* او المنهج *Method* يكون عاما ومتاح الوصول من قبل أي فصيل او دالة داخل او خارج الفصيل او الحزمة ***Package*** ويتم تعريفه كالتالي :

```
public  int x;                          تعريف متغير عام

public void printSomeThing()            تعريف منهج عام
{

}
```

2- المحدد خاص private :

يعني ان المتغير *Variable* او المنهج *Method* يكون خاصا ومتاح الوصول من قبل الفصيل نفسه ولا يمكن لأي فصيل اخر الوصول اليه وتغييره او استدعاءه حتى الموروث منه ويتم تعريفه كالتالي :

```
private  double  y;                     تعريف متغير خاص

private void myPrint()                  تعريف منهج خاص
{

}
```

3- المحدد محمي protected :

يعني ان المتغير *Variable* او المنهج *Method* يكون محميا ومتاح الوصول فقط من قبل أي فصيلة موجودة بنفس الحزمة (***package***) او أي فصيلة فرعية (***subclass***) موروثة من هذا الفصيل سواء كانت داخل او خارج الحزمة ويتم تعريفه كالتالي :

```
protected char  gender;                 تعريف متغير محمي

protected  int myAverage()              تعريف منهج محمي
{

}
```

3- المحدد الافتراضي default :

يعني عدم وضع أي محدد امام المتغير *Variable* او المنهج *Method* ويكون <u>متاح الوصول</u> <u>فقط من قبل أي فصيلة موجودة بنفس الحزمة</u> (***package***) ولا يمكن الوصول اليها من خارج الحزمة ويتم تعريفه كالتالي :

```
float  price;                          تعريف متغير افتراضي

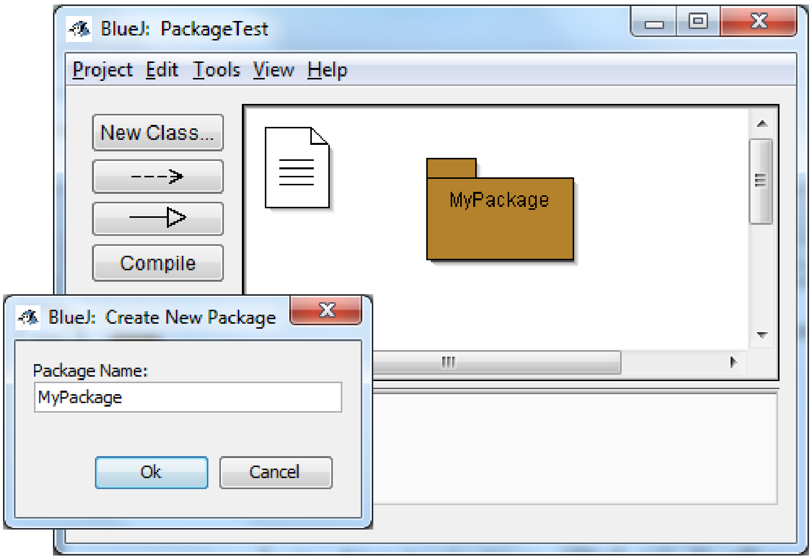
void printPrice()                      تعريف منهج افتراضي
{

}
```

<u>الحزمة Package :</u>

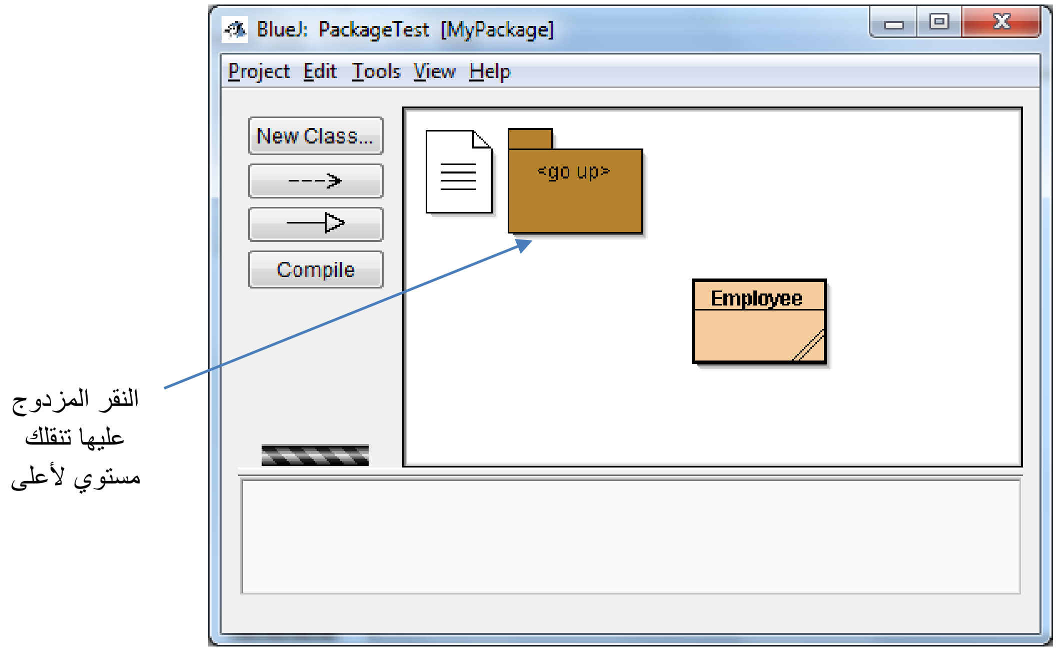
من احد افكار البرمجة الهدفية هي تجميع الفصائل ذات العلاقة مع بعضها البعض ووضعها في حزمة واحدة ، مثلا حزمة للرسومات واخرى للشبكات و هكذا لإعادة استخدامها مستقبلا.

لتكوين حزمة جديدة من داخل الـ blueJ نختار مشروع جديد ثم من قائمة تحرير Edit نختار حزمة جديدة New Package نعطي اسما لها وليكن MyPackage ليظهر شكلها كالتالي :



➢ انقر على *MyPackage* نقرا مزدوجا للدخول اليها وقم بإنشاء فصيل جديد داخلها وليكن فصيل الموظف ***Employee*** ، لاحظ ان اول سطر بها هو :

```
package MyPackage;
```



➢ ضمن فصيل الموظف (Employee) احذف كل البرمجة عدا الاتي :

```
package MyPackage;

public class Employee
{

    public Employee()
    {

    }

}
```

➢ ضمن فصيل عرف اربعة حقول هي :

- الاسم كحقل افتراضي.
- العمر كحقل عام.
- المرتب كحقل محمي.
- الوظيفة كحقل خاص.

ثم في الباني استقبل قيم تمثل الحقول وقم بإسنادها لهذه الحقول كالتالي :

```
package MyPackage;

public class Employee
{
    String name;
    public int age;
    protected double salary;
    private String job;

    public Employee(String n, int a, double s, String j)
    {
        name = n;
        age = a;
        salary = s;
        job = j;
    }
}
```

الان اضف دوال الـget لكل الحقول ليصبح شكل الفصيل كاملا كالتالي :

```
package MyPackage;

public class Employee
{
    String name;
    public int age;
    protected double salary;
    private String job;

    public Employee(String n, int a, double s, String j)
    {
        name = n;
        age = a;
        salary = s;
        job = j;
    }

    public String getName()
    {     return name;       }

    public int getAge()
    {     return age;    }

     public double getSalary()
    {     return salary;  }

     public String getJob()
    {     return job;    }
}
```

➢ ضمن حزمة ***MyPackage*** قم بإنشاء فصيل جديد داخلها وليكن فصيل الاختبار ***Test1*** ، لاحظ ان اول سطر بها هو نفس الاسم الدال على نفس الحزمة :

```
package MyPackage;
```

كون دالة رئيسية ***main*** وبداخلها كون كائن جديد اسمه ***e1*** من الفصيل ***Employee*** ، وقم بطباعة الاسم والدرجة والمرتب مستخدما اسماء الحقول مباشرة كالتالي :

```
package MyPackage;

public class Test1
{
    public static void main(String[] args)
    {
       System.out.print('\f');

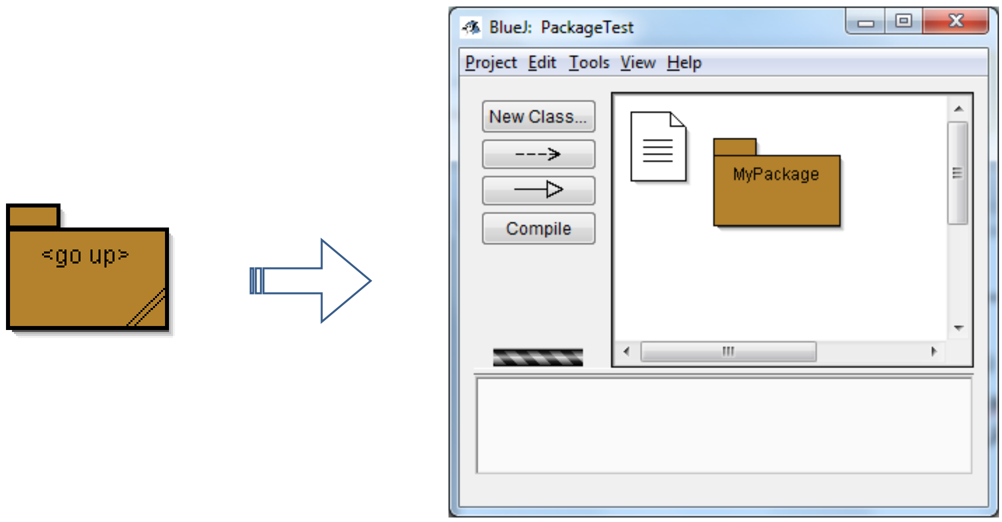
       Employee e = new Employee("Ali", 20, 800.0, "Manager");

       System.out.println("Name : " + e.name);
       System.out.println("Age : " + e.age);
       System.out.println("Salary : " + e.salary);

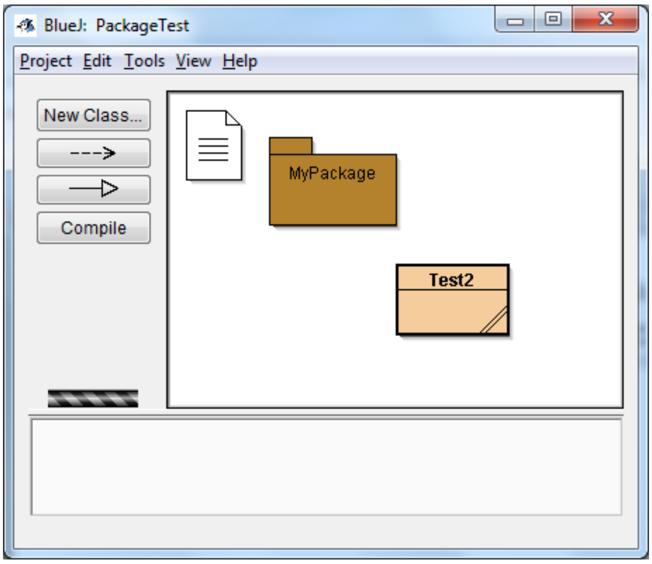
    }
}
```

قمنا باستخدام اسم الحقل مباشرة ***e.name*** لأن الوصول اليه افتراضي . كذلك الحال للعمر ***e.age*** لأنه عام ***public*** ، وايضا المرتب ***e.salary*** لأنه محمي ***protected*** ويمكن الوصول اليها من أي فصيل داخل نفس الحزمة ، ولا يمكن الوصول الى حقل الوظيفة ***job*** من خارج الفصيل لأنه حقل خاص ***private*** .

➢ الان انتقل الى الاعلى بالنقر المزدوج على ايقونة ***up go*** لتكون خارج حزمة ***MyPackage*** كالتالي :



➢ خارج الحزمة كون فصيل اختبار اخر اسمه ***Test2*** :



واكتب به الاتي :

```
Import MyPackage.Employee;

public class Test2
{

    public static void main(String[] args)
    {
        System.out.print('\f');

        Employee e2 = new Employee("Arm", 40, 400, "Teacher");

        System.out.println("Age : " + e2.age);

    }
}
```

لاحظ اننا نريد ان نستخدم فصيل الموظف ***Employee*** ضمن حزمة ***MyPackage*** ففي اول سطر قمنا باستيرادها كالتالي :

```
Import MyPackage.Employee;
```

او يمكن كتابتها كالتالي لاستراد كل الفصائل بالحزمة ***MyPackage*** :

```
Import MyPackage.*;
```

لاحظ انه يمكننا فقط الوصول الى حقل العمر ***age*** لأنه عام ***public*** اما باقي الحقول و هي الاسم والمرتب والوظيفة فمن غير المسموح به الوصول اليها ، لأن الوصول اليها اما ان يكون افتراضي او محمي او خاص ولان هذا الفصيل الجديد ***test2*** يقع خارج حزمة ***MyPackage*** فلا يمكن ان يصل اليها.

ملاحظة :

* لاحظ ان مستوى الوصول الافتراضي ***default*** هو نفسه مستوى الوصول المحمي ***protected*** في ان كلاهما يمكن الوصول اليهما من أي فصيل ضمن الحزمة ، عدا ان **المحمي** يمكن ان يتم الوصول اليه من أي فصيل ضمن الحزمة او أي فصيل **موروث** ضمن الحزمة او خارجها .

ملخص المحددات Modifiers :

مجال الرؤية	عام public	محمي protected	افتراضي default	خاص private
من داخل نفس الفصيل	✓	✓	✓	✓
من داخل فصيل في نفس الحزمة	✓	✓	✓	✗
من داخل فصيل خارج الحزمة	✓	✗	✗	✗
من داخل فصيل فرعي في نفس الحزمة	✓	✓	✓	✗
من داخل فصيل فرعي خارج الحزمة	✓	✓	✗	✗

أي أن :

العـــــام : يكون مرئي لكل الفصائل.

الافتراضي : يكون مرئي فقط للفصائل بنفس الحزمة.

المحمـــي : يكون مرئي للفصائل بنفس الحزمة والفصائل الفرعية خارج وداخل الحزمة.

الخـــاص : يكون مرئي فقط داخل نفس الفصيل.

الدوال الرياضية :

هناك العديد من الدوال الرياضية الجاهزة التي يمكن استخدامها مسبوقة باسم الفصيل Math كالتالي:

الدالة	وصفها	مثال	النتيجة
Math.sqrt(x)	لإيجاد الجدر التربيعي	Math.sqrt(25)	5
Math.pow(x, n)	لإيجاد قوة عدد x^n	Math.pow(4, 2)	16
Math.sin(x)	لإيجاد جيب الزاوية (جا)	Math.sin(30 * 3.14/180)	0.5
Math.cos(x)	لإيجاد جيب تمام الزاوية (جتا)	Math.cos(60 * 3.14/180)	0.5
Math.tan(x)	لإيجاد ظل الزاوية (ظا)	Math.tan(45 * 3.14/180)	1
Math.asin(x)	لإيجاد معكوس الجيب (جا)-1 قتا	Math.asin(30 * 3.14/180)	0.55
Math.acos(x)	لإيجاد معكوس جيب التمام (جتا)-1 قا	Math.acos(30 * 3.14/180)	1.02
Math.atan(x)	لإيجاد معكوس الظل (ظا)-1 ظتا	Math.atan(30 * 3.14/180)	0.48
Math.toRadians(x)	تغيير x من الدرجات الى التقدير الدائري (أي x*3.14/180)	Math.toRadians(30)	30*3.14/180
Math.toDegrees(x)	تغيير x من التقدير الدائري الى الدرجات (أي x*180/3.14)	Math.toDegrees (30)	30*180/3.14
Math.exp(x)	لإيجاد القيمة الاسية e	Math.exp(1)	2.718
Math.log(x)	لإيجاد اللوغاريتم الطبيعي	Math.log(2.718)	1
Math.round(x)	تقريب x لأقرب عدد صحيح	Math.round(3.6) Math.round(3.3)	4 3
Math.ceil(x)	التقريب لأقرب عدد صحيح اكبر من x	Math.ceil(3.1)	4
Math.floor(x)	التقريب لأقرب عدد صحيح اصغر من x	Math.floor(3.9)	3
Math.abs(x)	لإيجاد القيمة المطلقة	Math.abs(-20) Math.abs(20)	20
Math.random()	لتوليد رقم عشوائي كسري بين 0 واقل من 1	Math.random()	0.75393
Math.max(x, y)	ترجع القيمة الاكبر بين x و y	Math.max(5, 30)	30
Math.min(x, y)	ترجع القيمة الاصغر بين x و y	Math.min (5, 30)	5

كما يوجد ضمن الفصيل Math ثابت هو ط (π) وهو يساوي تقريبا 3.14 حيث يمكن استخدامه كالتالي :

```
Math.PI
```

ملاحظات :

القيم ضمن دوال الجيب (sin) وجيب التمام (cos) والظل (tan) يجب ان تكون بالتقدير الدائري وليس الدرجات ، لذلك يجب تحويلها الى التقدير الدائري بإحدى الطريقتين :

- ضرب الزاوية في $\frac{ط}{180}$
  فمثلا :

جا (30) ← `Math.sin(30 * 3.14/180)`

- استخدام دالة التحويل الى التقدير الدائري Math.toRadians() .

فمثلا :

جتا (60) ← `Math.cos ( Math.toRadians(60) )`

مثال أخر :

```
Math.toDegrees(Math.asin(Math.sin(Math.toRadians(30))))
```

النتيجة ستكون (30) :

حيث اولا وجدنا جيب الزاوية 30 بعد تحويلها الى التقدير الدائري ، ثم وجدنا معكوسها asin ومن ثم قمنا بتحويل النتيجة الى درجات باستخدام toDegrees().

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Max3Values*** يحتوي فقط على باني يستقبل ثلاثة قيم صحيحة ثم يطبع اعلى قيمة بين القيم الثلاثة.

```
public class Max3Values
{
    public Max3Values(int a, int b, int c)
    {
        int max =  Math.max( Math.max(a, b)   ,   c);

        System.out.print('\u000C');
        System.out.println("Maximum Value is : " + max);
    }
}
```

لاحظ اننا استخدمنا دالة max داخل نفسها لإيجاد اكبر قيمة بين اول عددين (a, b) ثم الاكبر منهما يقارن مع العدد الثالث (c) لإيجاد الاكبر بين الثلاثة :

Math.max(Math.max(a, b)   ,   c);

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***MyRandom*** يحتوي فقط على باني يستدعي دالة اسمها ()myRandom والتي ستقوم بترجيع عدد صحيح قيمته ما بين 0 الى اقل من 100 يكون مولد بشكل عشوائي.

```
public class MyRandom
{
    public MyRandom()
    {
       System.out.print('\f');
       System.out.println( "Random Number : " + myRandom() );
    }

    private int myRandom()
    {
        int r = (int)( Math.random() * 100  );
        return (r);
    }
}
```

لاحظ أن :

- اسم الفصيل هو MyRandom وأن اسم الدالة هي myRandom ، حيث هناك اختلاف في الحرف الاول احدها كبير والاخر صغير وهذا يجوز ولا خطا فيه.
- ان نوع محدد الدالة *()myRandom* هي خاص ***private*** لأننا سنقوم باستخدامها فقط داخل الفصيل ، كما يمكننا كتابتها عامة public والنتيجة ستكون واحدة.
- الرقم العشوائي المولد بواسطة *()Math.random* سيكون بين (0 و 0.999) لذلك قمنا بضربها في 100 والتي سينتج عنها قيمة كسرية بين 0 الى 99.999 ، ومن ثم قمنا بتحويلها قسريا الى قيمة صحيحة باستخدام (int) حيث ستقوم بحذف الكسر.

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Equation*** يحتوي فقط على باني يستدعي دالة اسمها myEquation(x, y) والتي ستقوم بحساب وطباعة قيمة w بتنسيق خانتين بعد الكسر:

$$w = \sqrt[4]{\sin(x)} - \cos(y) \quad + x^3$$

```
public class Equation
{

   public Equation(int x, int y)
   {
      myEquation(x, y);         استدعاء الدالة مع تمرير قيم لها
   }

   public void myEquation(int x, int y)
   {
     double w = Math.pow( Math.sin(Math.toRadians(x)), 0.25)
                - Math.cos(y * 3.14/180) + Math.pow(x,3);


     System.out.print('\u000C');

     System.out.format("w = %6.2f", w);
    }
}
```

لاحظ أن :

- تم استقبال قيمتين ضمن الباني x و y وتم تمريرها الى الدالة ***myEquation(x, y)*** .
- حيث تم استقبال القيمتين المرسلتين ضمن الدالة والتي سوف لن ترجع قيمة (***void***) ومن ثم حساب المعادلة كالتالي :

وحيث ان : $\sqrt[4]{\sin(x)}$ يعني $(\sin(x))^{0.25}$

فان :

$\sqrt[4]{\sin(x)}$ ➔ `Math.pow( Math.sin(Math.toRadians(x)), 0.25)`

$\cos(y)$ ➔ `Math.cos(y * 3.14/180)` او `Math.cos(Math.toRadians(y))`

x^3 ➔ `Math.pow(x,3)`

- تم طباعة النتيجة باستخدام System.out.format() والتي تقوم بنفس عمل System.out.printf() من حيث العمل.

ايضا يمكن اعادة كتابة الدالة myEquation() لتنسيق المخرجات كالتالي :

```
public void myEquation(int x, int y)
{
    double w = Math.pow( Math.sin(Math.toRadians(x)), 0.25)
                - Math.cos(y * 3.14/180) + Math.pow(x,3);

      System.out.print('\u000C');

      String f = String.format("The Result = %7.2f" , w);
      System.out.println(f);
}
```

لاحظ اننا قمنا بتنسيق النتيجة بتعريف سلسلة ***f*** ثم استخدمنا دالة التنسيق *format* ضمن الفصيل ***String*** وهي تأخذ نفس معاملات التحويل في دالة *printf()* والتي سبق شرحها حيث ان %7.2f يعني حجز 7 خانات للعدد الحقيقي منها خانتين كسريتين . ثم قمنا بطباعة السلسلة ***f*** منسقة باستخدام جملة الطباعة *println()*.

مثال :

مستخدما مفاهيم البرمجة الشيئية ، اكتب برنامج جافا لتكوين فصيل اسمه ***Circle*** والذي يحتوي على باني واحد فقط يستقبل قيمة كسرية تمثل نصف قطر دائرة ، كما يحتوي على حقل واحد فقط وهو نصف القطر ومن ثم تكوين منهجين عاميين واحد لحساب مساحة الدائرة والاخر لحساب محيطها.

```
public class Circle
{
    private double radius;

    public Circle(double rd)
    {
        radius = rd;
    }
    public double getRadius()
    {
        return radius;
    }
    public double getArea()
    {
        return (Math.PI * radius * radius);
    }
    public double getCircumference()
    {
       return (2 * Math.PI * radius);
    }
}
```

التحويل بين الأنواع Casting type values :

يعني تحويل نوع من البيانات الى نوع اخر قسريا فمثلا:

```
float f = 10.9f;
int   i = f;
```

في هذه العملية قمنا بتعريف متغير حقيقي f وخصصنا له قيمة 10.9 ، ثم قمنا بتعريف متغير صحيح i وخصصنا له قيمة المتغير الحقيقي f . مباشرة عند تنفيذ البرنامج سوف تظهر رسالة خطأ تبن انه لا يمكن تخصيص قيمة كسرية الى متغير صحيح. لذلك يمكننا علاج هذا الوضع بالتحويل القسري للنوع وذلك بذكر النوع بين قوسين قبل القيمة المخصصة كالتالي :

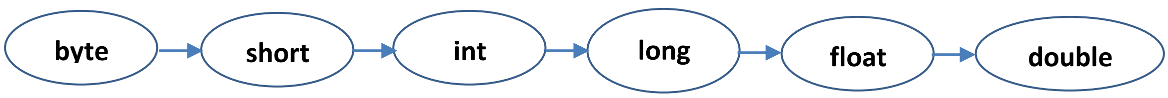
```
        (type)  value;
i   =   (int)    f ;
```

الان سيتم حذف الكسر مهما كانت قيمته واسناد فقط الجزء الصحيح الى المتغير i (أي ان i الان تساوي 10)

القاعدة هي : عند التحويل من الاصغر الى الاكبر فالمترجم سوف لن يعترض عليه فمثلا :

```
int i = 30;
float a = i;
```

هذا الامر صحيح ولا غبار عليه ولكن بالعكس يجب عليك ان تحول النوع قسريا كما سبق ذكره وذلك كالتالي :



مثال اخر :

```
double d = 40.98;
byte    i = (byte) d;     ///    the Result 40
```

كما يمكن التحويل أوتوماتيكيا من النوع الرمزي الى النوع الصحيح فمثلا :

```
char c = 'A';
int  i = c;     ///    now i = 65

System.out.print(i);
```

نتيجة الطباعة هي 65 وذلك لان الرمز 'A' ضمن جدول الاسكي يساوي 65.

أما لتحويل من النوع الصحيح الى الرمزي فيجب التحويل قسريا كالتالي :

```
int  i = 65;
char c =  (char) i;      ///   now c = 'A'

System.out.print(c);
```

عند الطباعة فان النتيجة ستكون 'A' .

أما التحويل من عدد صحيح الى سلسلة رمزية فهناك طريقتين :

- اما ان تضيف رقم الى سلسلة رمزية فارغة باستخدام علامة + كالتالي :

```
int  i = 123;
String s = "" + i;
```

وبالتالي فان s الان ستحتوي على "123" كسلسلة رمزية وليس كعدد.

- واما باستخدام الدالة **tostring()** والخاصة بالفصيل **Integer** كالتالي :

```
int    i = 123;
String s = Integer.toString(i);
```

حيث قامت الدالة *toString(i)* بتحويل الرقم i الى سلسلة "123" ومن ثم تخصيصها الى المتغير s .

أما التحويل من سلسلة رمزية الى عدد صحيح فيتم ذلك باستخدام الدالة parseInt() التابعة للفصيل Integer كالتالي :

```
String s = "456";

int    i = Integer.parseInt(s);
```

الان المتغير i سيحتوي على القيمة 456 كعدد.

كذلك الحال بالنسبة للتحويل من سلسلة رمزية الى عدد حقيقي :

```
float  f = Float.parseFloat("123.4");
```

أو

```
double d = Double.parseDouble("123.4");
```

فصائل تغليف الانواع الاساسية والتغليف الالي :
Primitive wrapper calss and auto-boxing

لكون الارقام ليست كائنات ضمن الجافا لذلك تم تكوين فصيل لكل نوع اساسي ضمن الجافا كالتالي :

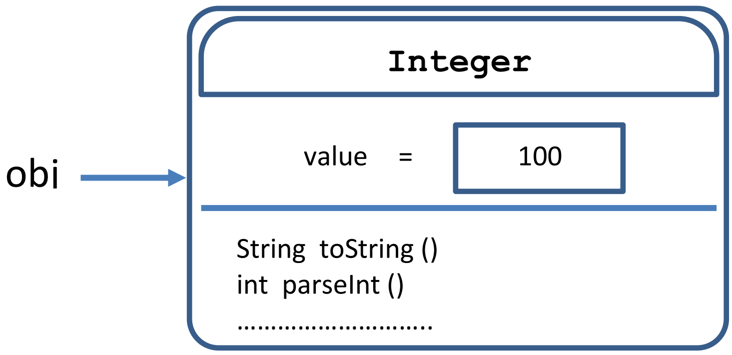
النوع الاساسي Primitive type	الفصيل المغلف Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

لاحظ ان كل انواع الفصائل تبدا بحرف كبير ، وابتداء من جافا 5 يتم التحويل بين الانواع الاساسية والفصائل المقابلة لها أوتوماتيكيا . فعملية التحويل بين النوع الاساسي (int مثلا) ونوع الفصيل (Integer) تسمى *auto-boxing* . اما عند التحويل من نوع الفصيل الى النوع الأساسي فتسمى *auto-unboxing* .

فلتعريف كائن صحيح من نوع الفصيل Integer نكتب الاتي :

```
Integer obj = new Integer(100);
```

هنا قمنا بتكوين كائن صحيح اسمه obj يحتوي على قيمة 100 كالتالي :



هذا الكائن الجديد يحتوي على العديد من المناهج التي يمكننا الاستفادة منها كتحويل النوع من عدد صحيح الى سلسلة *tostring()* وتحويل السلسلة الى عدد صحيح *parseInt()* وغيرها اضافة الى اننا نستطيع معاملتها معاملة النوع الاساسي فمثلا اذا كان لدينا كائنان صحيحان (obj1 و obj2) ومتغيران صحيحان (i و j) فيمكننا القيام بعمليات التخصيص التالية :

```
public Test-AutoBoxing()
{
    Integer obj1 = new Integer(100);
    Integer obj2;

    int i = 20;
    int j;

    obj2 = i;          //// auto-boxing
    j = obj1;          //// auto-unboxing

    System.out.print('\f');

    System.out.println("obj 1 : " + obj1);
    System.out.println("obj 2 : " + obj2);
    System.out.println("i : " + i);
    System.out.println("j : " + j);
}
```

لاحظ اننا قمنا بإسناد قيمة ***i*** الى الكائن ***obj2*** وستقوم الجافا بعملية التحويل أوتوماتيكيا (*autoboxing*) ، ايضا قمنا بإسناد قيمة الكائن ***obj1*** الى المتغير ***j*** وسيقوم الجافا بعملية التحويل أوتوماتيكيا (*auto-unboxing*). والنتيجة ستكون كالتالي :

```
obj1 : 100
obj2 : 20
i : 20
j : 100
```

قبل جافا 5 كان عملية التحويل من النوع الفصيل الى النوع الاساسي تتم باستخدام دالة تحويل كالتالي :

```
Integer obj = new integer(200);
int j = obj.intValue();
```

لاحظ استخدامنا للدالة ***intValue()*** لتحويل قيمة من نوع الفصيل الصحيح الى عدد صحيح اساسي وكذلك الحال : بحيث يمكننا استخدام ***doubleValue()*** و ***booleanValue()*** وهكذا.

اما الان فلا داعي لاستخدام مثل هذه الدول لان التحويل يتم أوتوماتيكيا كما شاهدنا سابقا.

دوال السلاسل الرمزية String Methods:

يمكننا تعريف سلسلة رمزية كالتالي :

```
String s1 ;
```

او يمكننا اعطاءها قيمة مبدئية اثناء التعريف كالتالي :

```
String s2 = "Computer";
String s3 = "Computer";
```

هنا السلسلتين s1 وs2 يمكننا معاملتها كنوع اساسي بحيث s1 تكون مساوية s2 أي أن :

```
System.out.print(s2==s3);        ////  true
```

النتيجة ستكون صحيحة true .

ولكون الـ String (حرف S كبير) فصيل فيمكننا تكوين كائنات منه كالتالي :

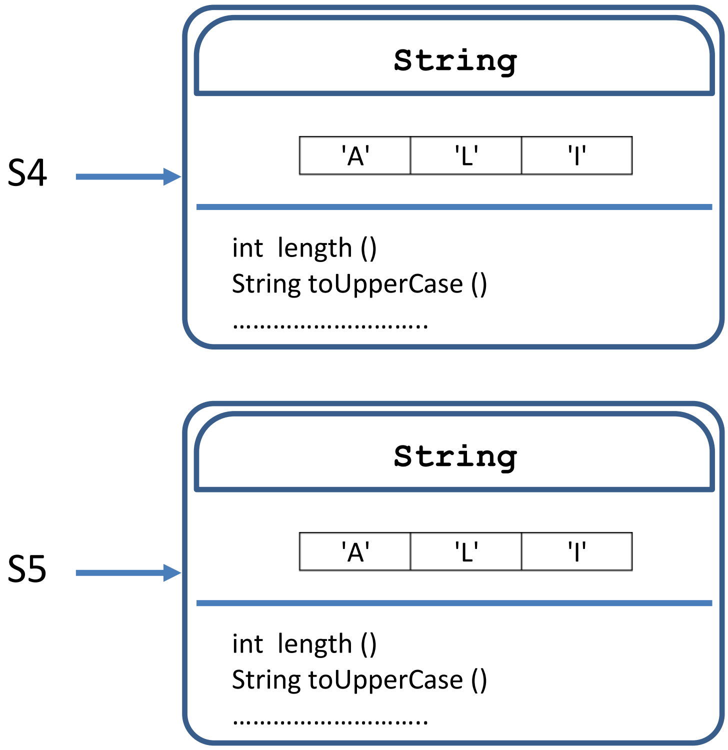
```
String s4= new String ("ALI");
String s5= new String ("ALI");
```

ماذا ستكون النتيجة من جملة الطباعة التالية :

```
System.out.print(s4==s5);     //////   false
```

النتيجة كانت خطا False فلماذا ؟؟؟

لأننا قمنا بتكوين كائنين مختلفين هما s4 و s5 كالتالي :



ولكن محتويات s4 هي مساوية لمحتويات s5 ، لذلك يمكننا استخدام المنهج ***equals()*** لمعرفة تساوي قيم السلاسل كالتالي :

```
System.out.print( s4.equals(s5) );     //////   True
```

النتيجة الان ستكون صحيحة true ، حيث *equals()* تقارن محتويات s4 (وهي هنا "*ALI*") مع محتويات s5 (وهي هنا "*ALI*") ولا تقارن قيمة المؤشرات s4 مع s5 .

ايضا يمكننا استخدام دالة ***equalsIgnoreCase()*** (المساواة مع اهمال حالة الاحرف) والتابع لفصيل الـ*String* لمقارنة مساواة كائنين من نوع السلسلة مع اهمال حالة الاحرف كونها حروف كبيرة او صغيرة فمثلا :

```
String s6 = new String("COMPUTER");
String s7 = new String("computer");
System.out.println("Result = " + s6.equalsIgnoreCase(s7)  );
```

هنا النتيجة ستكون true ، حيث قارن السلسلتين بغض النظر عن حالة الاحرف هل هي كبيرة او صغيرة.

مع الملاحظ أنه يمكننا معاملة السلسلة الرمزية كمصفوفة من الرموز فمثلا الاعلان التالي :

```
String s8= new String ("Computer");
```

يمكننا تمثيله بالشكل الاتي :

s8

'C'	'o'	'm'	'p'	'u'	't'	'e'	'r'
0	1	2	3	4	5	6	7

فصيل الـString يحتوي على العديد من الدوال والتي يمكننا استخدامها في معالجة السلاسل الرمزية ،ة حيث سنقوم بدراسة بعض منها فقط كالتالي :.

➢ **<u>دالة معرفة طول السلسلة الرمزية length() :</u>**

تقوم هذه الدالة بإيجاد طول السلسة الرمزية فمثلا :

```
String s = "Computer";
int    i = s.length();
```

هنا ***i*** تحتوي على طول السلسلة ***s*** وهي هنا 8 ،كما يمكننا طباعة الطول ضمن دالة الطباعة مباشرة كالتالي :

```
System.out.println("Result = " +  s.length()  );
```

➢ دالة سلسلتين في سلسلة رمزية واحدة concat() :

تقوم هذه الدالة بوصل (concatenation) سلسلتين رمزيتين فمثلا :

```
String s1 = "My ";
String s2 = "Computer";
System.out.println("Result = " +  s1.concat(s2)  );
```

هنا النتيجة ستكون `"My Computer"` . ايضا يمكننا استخدام علامة (+) لوصل سلسلتين او اكثر كالتالي :

```
String s1 = "This" + "  " + "c" + "  " + "a concatenation";
```

النتيجة ستكون :

```
"This is a concatenation"
```

➢ دالة الحصول على رمز ضمن سلسلة رمزية charAt() :

تقوم هذه الدالة باسترجاع رمز ضمن سلسله رمزية فمثلا اذا كانت :

```
String s = "Computer";
```

والذي يمكن تمثيلها بالشكل التالي :

s	'C'	'o'	'm'	'p'	'u'	't'	'e'	'r'
	0	1	2	3	4	5	6	7

فان :

```
Char c = s.charAt(3);
```

سوف يقوم باسترجاع الرمز 'p' ذو الموقع 3 .

مع ملاحظة ان اول رمز يأخذ القيمة 0 واخر رمز يأخذ القيمة (طول السلسلة - 1)

➢ دالة استبدال رمز مكان رمز اخر replace() :

تقوم هذه الدالة باستبدال رمز مكان اخر ضمن سلسلة رمزية فمثلا :

```
String s = "Computer";

System.out.println("Result = " + s.replace('e','*'));
```

أي قم باستبدال الرمز 'e' وضع مكانه الرمز '*' ، فالنتيجة ستكون : "comput*r"

➢ دالة تحويل السلسلة الى حروف كبيرة او صغيرة toUpperCase() & toLowerCase()

تقوم دالة ***toUpperCase()*** بتحويل رموز السلسلة الرمزية الى رموز كبيرة ، وتقوم الدالة ***toLowerCase()*** بتحويل رموز السلسلة الى رموز صغيرة فمثلا :

```
String s1 = "car";
System.out.println("Result = " + s.toUpperCase() );  /// CAR


String s2 = "ALI";
System.out.println("Result = " + s.toLowerCase() );  /// ali
```

الاولى ستقوم بطباعة "CAR" في حالة الاحرف الكبيرة ، والثانية ستقوم بطباعة "ali" في حالة الاحرف الصغيرة.

➢ دالة مقارنة سلسلتين compareTo() :

تقوم هذه الدالة بمقارنة سلسلتين حسب ترتيب الرموز ضمن جدول الاسكي وهي مفيدة عند ترتيب قائمة من الاسماء ابجديا حيث ترجع عدد صحيح قد يكون :

- اكبر من صفر اذا كان السلسلة الاولى اكبر من الثانية .
- تساوي صفر اذا كانت السلسلتين متساويتين.
- اصغر من الصفر اذا كانت السلسلة الاولى اصغر من الثانية (أي الاولى تقع قبل الثانية في الترتيب الابجدي) مع العلم بان الحروف الابجدية الصغيرة لها ارقام ضمن جدول الاسكي اكبر من الحروف الابجدية الكبيرة.

فمثلا :

```
String s = "Computer";
System.out.println(  s.compareTo("computer") );   /// -32
```

النتيجة ستكون (32-) لان الاولى تبدا بحرف كبير والثانية بحرف صغير.

➢ دالة استرجاع سلسلة فرعية substring() :

تقوم هذه الدالة بترجيع سلسلة فرعية من سلسلة اخر ابتداء من موقع معين الى موضع اخر كالتالي :

```
String s = "Manager";
System.out.println( s.substring(3,6) );    ///   age
```

s

'M'	'a'	'n'	**'a'**	**'g'**	**'e'**	'r'
0	1	2	3	4	5	6

أي استرجع سلسلة فرعية ابتداء من الموقع رقم 3 الى الموقع قبل الـ6 وبالتالي سيرجع "age" .

أما اذا حددنا فقط موضع البداية ولم نحدد موضع النهاية فسيرجع سلسلة فرعية ابتداء من موضع البداية الى نهاية السلسلة كالتالي :

```
System.out.println( s.substring(2) );    ///   nager
```

s

'M'	'a'	**'n'**	**'a'**	**'g'**	**'e'**	**'r'**
0	1	2	3	4	5	6

فالنتيجة ستكون "***nager***".

➢ دالة اول ظهور للرمز او السلسلة indexOf() :

تقوم هذه الدالة بترجيع عدد صحيح يمثل اول ظهور للرمز او السلسلة ضمن سلسلة اخرى ، اما في حالة عدم وجوده فسيرجع بـ 1- كالتالي :

```
String s = "Manager";
System.out.println("Result = " + s.indexOf('a') );   //// 1
```

حيث ترجع الرقم 1 وهو موقع اول ظهور للرمز 'a' ضمن السلسلة s . ايضا يمكن كتابتها :

```
String s = "Manager";
System.out.println("Result = " + s.indexOf("age") );   //// 3
```

حيث ترجع الرقم 3 وهو موقع اول ظهور للسلسلة الرمزية "age" ضمن السلسلة s .

قراءة المدخلات من لوحة المفاتيح Reading Input:

حتى الان قمنا بتعيين قيم للحقول اما بتخصيصها من داخل البرنامج او بتمرير معاملات الى المناهج باستخدام الـ*bluej* ، كما كنا نستخدم الكائن System.out لطباعة المخرجات على الشاشة. وهناك اكثر من طريقة للقراءة من لوحة المفاتيح.

انه بالإمكان استخدام الكائن **System.in** لقراءة المدخلات من لوحة المفاتيح ولكن للقيام بذلك علينا اولا تعريف كائن من الفصيل **Scanner** والموجود ضمن الحزمة **java.util** . هذا الفصيل يزودنا بالعديد من المناهج التي تمكننا من قراءة اعدد صحيحة او حقيقية او منطقية او سلاسل رمزية من لوحة المفاتيح ، كما يمكن استخدامه ايضا للقراءة من الملفات النصية.

لاستخدامه علينا اولا استيراد الفصيل **Scanner** كما يلي :

```
import java.util.Scanner;
```

او يمكننا استيراد كل فصائل الـ **java.util** باستخدام * كالتالي :

```
import java.util.*;
```

ثم علينا تكوين كائن وليكن (kb) يمثل لوحة المفاتيح من فصيل الـScanner كالتالي :

```
Scanner kb = new Scanner(System.in);
```

حيث ان الـSystem.in تمثل لوحة المفاتيح .

الان يمكننا القراءة من مدخلات لوحة المفاتيح باستخدام احدى المناهج التالية :

الدالة	الوظيفة
`nextInt()`	لقراءة عدد صحيح
`nextDouble()`	لقراءة عدد حقيقي
`nextLine()`	لقراءة سلسلة رموز
`next().charAt(0)`	لقراءة رمز واحد (اول رمز يتم ادخاله)
`nextBoolean()`	لقراءة قيمة منطقية (true او false)

فمثلا :

لقراءة عدد صحيح	`int a = kb.nextInt();`
لقراءة عدد حقيقي	`double b = kb.nextDouble();`
لقراءة سطر كامل من الرموز	`String c = kb.nextLine();`
لقراءة رمز واحد فقط	`char d = kb.next().charAt(0);`
لقراءة قيمة منطقية	`boolean e = kb.nextBoolean();`

لاحظ أن المنهج **nextLine()** يقوم بقراءة سطر كامل من الرموز بما فيها الفراغات حتى الضغط على مفتاح الادخال *Enter* . أما المنهج **next()** فيقوم بقراءة اول كلمة أي سيقوم انهاء الادخال عند الوصول الى اول فراغ.

ايضا لا يوجد منهج محدد لقراءة قيمة رمزية *char* (رمز واحد فقط) لذلك نستخدم احدى دوال معالجة السلاسل وهو **chatAt(0)** لقراءة اول رمز يتم ادخاله (الرمز الواقع في الموقع 0).

يفضل داخل قبل استقبال قيمة من المستخدم اظهار رسالة تبين له ما المطلوب ادخاله فمثلا :

```
System.out.print("Enter Integer Number : ");
int a = kb.nextInt();
```

حيث تظهر الرسالة اولا ثم يقوم المستخدم بإدخال قيمة صحيحة.

مثال :

اكتب برنامج جافا لتكوين فصيل اسمه ***MyInputs*** والذي يحتوي على الدالة الرئيسية main ومن ثم يقوم بقراءة سلسلة رمزية ومن ثم رمز ثم قيمة صحيحة ثم قيمة حقيقية ثم قيمة منطقية وطباعتها.

```
import java.util.Scanner;
public class MyInputs
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

        System.out.print("Enter String : ");
        String s = kb.nextLine();

        System.out.print("Enter Charater : ");
        char c = kb.next().charAt(0);

        System.out.print("Enter Integer Number : ");
        int i = kb.nextInt();

        System.out.print("Enter double Number : ");
        double d = kb.nextDouble();

        System.out.print("Enter Boolean : ");
        boolean b = kb.nextBoolean();

        System.out.println("String is : " + s);
        System.out.println("Characher is : " + c);
        System.out.println("Integer is : " + i);
        System.out.println("Double is : " + d);
        System.out.println("Boolean is : " + b);
    }
}
```

مثال :

اكتب برنامج جافا يقرأ من لوحة المفاتيح عددين صحيحين هما x و n ومن ثم حساب :

$$Y = X^n$$

```
import java.util.Scanner;
public class Power
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

        System.out.print("Enter X & n : ");

        int x = kb.nextInt();
        int n = kb.nextInt();

        System.out.format(" Y = %.2f", Math.pow(x, n) );
    }
}
```

مثال :

اكتب برنامج جافا يتكون من فصيلين الاول اسمه خط الانتاج **Line** والذي يحتوي على حقل واحد هو الايراد ***income*** ، ويحتوي على باني واحد يستقبل قيمة الايراد لتخزينها في الحقل . يحتوي هذا الفصيل ايضا على ثلاث مناهج الاول يرجع قيمة الايراد والثاني يحسب ويرجع قيمة الضريبة والثالث يحسب ويرجع قيمة صافي الايراد بعد خصم الضريبة. الضريبة غير معروفة يتم تمريرها كبارامتير الى منهجي حساب الضريبة وحساب الصافي. اما الفصيل الثاني فاسمه المصنع **Factory** والذي يحتوي على الدالة الرئيسية *main()* و تقوم بقراءة قيمتين من الشاشة الاول تمثل ايراد الخط والثاني نسبة الضريبة على الايراد ومن ثم تكوين كائن من فصيل خط الانتاج وتمرير قيمة الايراد له ، ثم طباعة قيمة الضريبة وصافي الدخل على الشاشة.

```
public class Line
{
    private double income;

    public Line(double i)
    {       income = i;          }

    public double getIncome()
    {       return income;       }

    public double calcTax(double taxRate)
    {
        return (taxRate * income);
    }

    public double calcNet(double taxRate)
    {
        return (income - calcTax(taxRate) );
    }
}
```

دالة حساب الضريبة

دالة حساب الصافي

تقوم باستدعاء دالة حساب الضريبة من ثم طرحها من الايراد لإيجاد الصافي

```
import java.util.Scanner;

public class Factory
{
  public static void main(String[] args)
  {
    Scanner kb = new Scanner(System.in);
    System.out.print('\u000C');

    System.out.print("Enter Incomes&tax rate for Line1 : ");

    double income1 = kb.nextDouble();
    double tax1 = kb.nextDouble();

    Line line1 = new Line(income1);

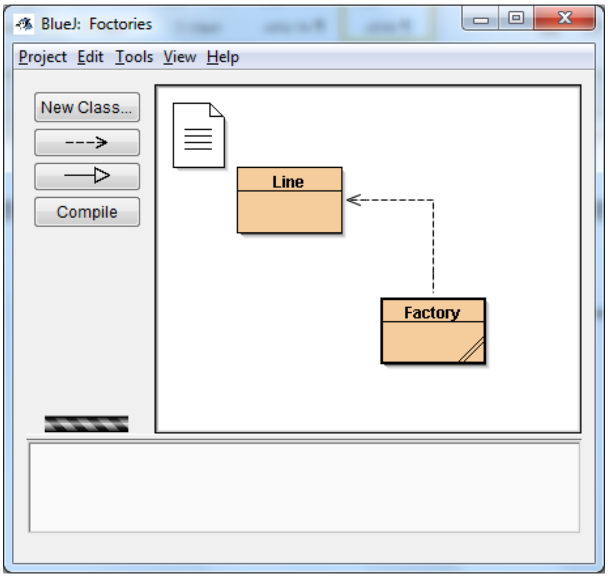
    System.out.println("Tax Line1 : " + line1.calcTax(tax1) );
    System.out.println("Net Line1 : " + line1.calcNet(tax1) );
  }
}
```

قراءة الايراد ونسبة الضريبة من لوحة المفاتيح

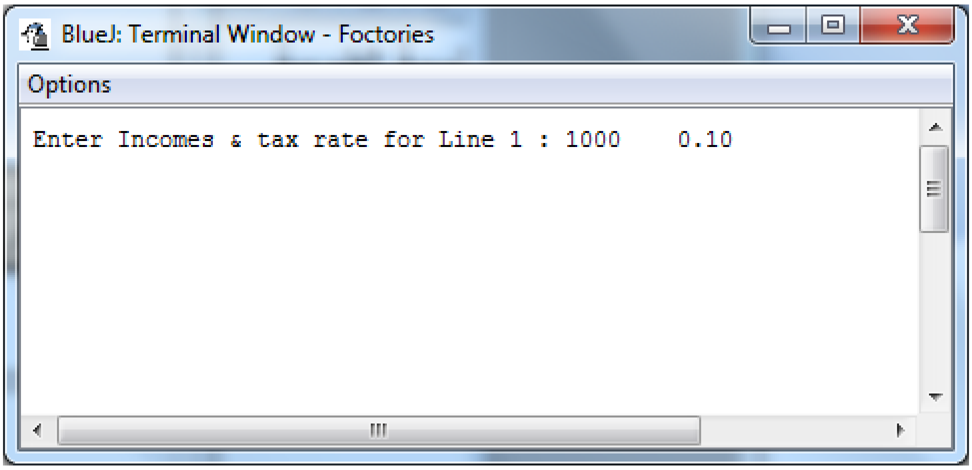
تكوين خط انتاج جديد

طباعة الضريبة والصافي

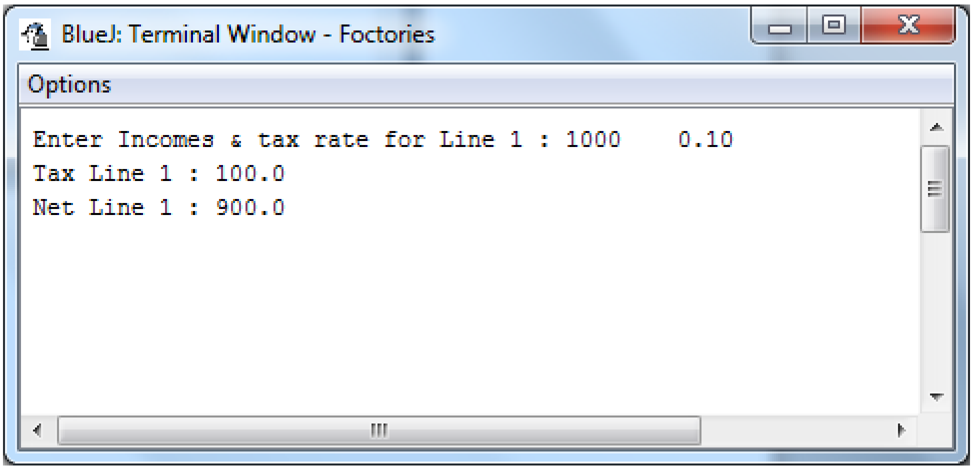
عند تكوين الفصائل :



عند تنفيذ دالة الـmain() من فصيل الـFactory وادخال قيمتين اولها الايراد (هنا 1000) والثانية هي نسبة الضريبة (هنا 10% حيث تكتب 0.10) :



بعد النقر على الـ*Enter* واظهار النتائج :



تدريب (3)

1- اكتب برنامج يتكون من فصيلين :

الفصيل الاول اسمه ***Operations*** يحتوي على ثلاثة حقول كسرية خاصة هي a و b و Result .

كما يحتوي على عدد 2 باني : **الاول** بدون معاملات ويخصص 0 للحقول الثلاثة ، والباني الثاني يستقبل عدد معاملان كسريان لتخزينها في الحقلين a و b .

كما يحتوي الفصيل الاول على عدد خمسة مناهج عامة :

الاول اسمه ()addition بدون معاملات ويرجع حاصل جمع العددين a و b.

الثاني اسمه ()subtraction بدون معاملات ويرجع حاصل طرح العددين a و b.

الثالث اسمه ()multiplication بدون معاملات ويرجع حاصل ضرب العددين a و b.

الرابع اسمه ()division بدون معاملات ويرجع حاصل قسمة العددين a و b.

الخامس اسمه ()getResult بدون معاملات ويرجع اخر عملية تم القيام بها.

اما الفصيل الثاني اسمه ***UseOperations*** فيحتوي على الدالة الرئيسية main ويقوم بتكوين كائن من الفصيل الاول مع إرسال قيم للحقيلين a و b ضمن الباني ثم استدعاء العمليات الحسابية الاربعة ضمن جملة الطباعة ومن ثم استدعاء المنهج *getResult()* لمعرفة نتيجة اخر عملية تم القيام بها.

2- كون مشروع جديد يحتوي على فصيلين ، الاول اسمه ***Calculator*** لا يحتوي على اية حقول ولا يحتوي على باني ، فقط يحتوي على منهج واحد اسمه ***calculate*** يستقبل عدد كسري واشارة رمزية ورقم كسري اخر ، ومستخدما جملة *switch* احسب احدى العمليات الحسابية الاربعة اعتمادا على الاشارة الرمزية المرسلة واظهر الرسالة المناسبة في حال ما كانت الاشارة الرمزية لا تمثل احدى الاشارات (+ ، - ، * ، /) . أما الفصيل الثاني اسمه ***UseCalculator*** ويحتوي على منهج main ويستخدم الفصيل الاول.

3- كون مشروع جديد يحتوي على فصيل واحد اسمه ***MaxAndMin*** يحتوي على ثلاثة حقول ويحتوي على باني واحد يستقبل قيمة واحدة صحيحة ويحتوي على ثلاثة مناهج عامة اولها store() لتخزين قيمة جديدة ، والمنهج الاخر fetch() لاسترجاع اخر قيم تم تخزينها ، والمنهج الثالث printValues() لطباعة اكبر واصغر واخر قيمة مدخلة. هذا المشروع يجب ان يحتفظ بسجل تاريخي للقيم التي تم تخزينها ، بمعنى : يجب ان تكون هناك امكانية للاحتفاظ واسترجاع اعلى واصغر قيمة تم ادخالها ضمن كل القيم المدخلة بالإضافة الى اخر قيمة مدخلة.

إجابة تدريب (3)

1- الفصيل الاول Operations :

```
public class Operations
{
    private double a, b, result;

    public Operations()
    {
        a = b = result = 0;
    }

    public Operations(double x, double y)
    {
         a = x;
         b = y;
         result = 0 ;
    }
    public double addition()
    {
         result = a + b;
         return result;
    }

    public double subtraction()
    {
         result = a - b;
         return result;
    }

    public double multiplication()
    {
         result = a * b;
         return result;
    }

    public double division()
    {
         result = a / b;
         return result;
    }

    public double getResult()
    {
         return result;
    }
}
```

تابع تدريب (1-3) الفصيل الثاني UseOperations :

```
public class UseOperations
{

  public static void main(String[] args)
  {

      System.out.println('\u000C');

      Operations op = new Operations(60, 20);

      System.out.println("Addition = " + op.addition() );
      System.out.println("Subtraction = " + op.subtraction() );
      System.out.println("Multiplication = " + op.multiplication() );
      System.out.println("Division = " + op.division() );

      System.out.println("The Last Operation = " + op.getResult() );

    }
}
```

2- الفصيل الاول Calculator :

```
public class Calculator
{

    public double calculate(double a, char op, double b)
    {
       double result = 0.0;

       switch (op)
       {
          case '+' : result = a + b;
                     break;

          case '-' : result = a - b;
                     break;

          case '*' : result = a * b;
                     break;

          case '/' : result = a / b;
                     break;
          default  : System.out.println("Invalid Operator !");
        }

        return result;
    }
}
```

تابع تدريب (3-2) الفصيل الثاني UseCalculator :

```
public class UseCalculator
{
  public static void main(String[] args)
  {
     Calculator myCalc = new Calculator();

     System.out.println('\f');

     System.out.println(" Sum = " + myCalc.calculate(10, '+', 5) );
     System.out.println(" Sub = " + myCalc.calculate(10, '-', 3) );
     System.out.println(" Mlti = " + myCalc.calculate(10, '*', 4) );
     System.out.println(" Div = " + myCalc.calculate(10, '/', 2) );
     System.out.println(" Other = " + myCalc.calculate(10, '$', 5) );
  }
}
```

3- الفصيل MaxAndMin :

```
public class MaxAndMin
{
     private int value;
     private int maxValue;
     private int minValue;

    public MaxAndMin(int n)
    {
         value = n;
         maxValue = n;
         minValue = n;
    }

    public void store(int v)
    {
        value = v;
        maxValue = Math.max(v, maxValue);
        minValue = Math.min(v, minValue);
    }

    public int fetch()
    {
         return value;
    }

    public void printValues()
    {
        System.out.print('\u000C');
        System.out.println("value : " + value + "\nMax : " +
                                   maxValue + "\nMin : " + minValue);
    }

}
```

تمرين (1)

1- اكتب برنامج جافا مكون من فصيل اسمه Max4Values يحتوي فقط على باني يستقبل اربعة قيم صحيحة ثم يطبع اعلى قيمة بين القيم الاربعة.

2- اكتب برنامج جافا مكون من فصيل اسمه Equation يحتوي فقط على باني يستدعي دالة اسمها clacArea(x, y) والتي ستقوم بحساب وطباعة قيمة A بتنسيق خانتين بعد الكسر.

$$A = \frac{1}{2}\sqrt{x^2y^2 - (x-y)^2}$$

3- مستخدما مفاهيم البرمجة الشيئية ، اكتب برنامج جافا لتكوين فصيل اسمه المستطيل Rectangle والذي يحتوي على باني واحد فقط يستقبل قيمتين كسريتين تمثل الطول والعرض ، كما يحتوي على حقلين وهو الطول والعرض ومن ثم تكوين مناهج الـget للحقول ومنهج محمي اسمه getArea() لحساب مساحة المستطيل.

4- مستخدما مفاهيم البرمجة الشيئية ، اكتب برنامج جافا لتكوين فصيل اسمه الاسطوانة Cylinder والذي يحتوي على باني واحد فقط يستقبل قيمتين كسريتين تمثل الاول نصف قطر القاعدة والثاني يمثل الارتفاع ، كما يحتوي على حقلين وهو نصف قطر القاعدة والارتفاع ومن ثم تكوين مناهج الـget للحقول ومنهج عام اسمه getVolume() لحساب حجم الاسطوانة حيث :

حجم الاسطوانة = ط نق^2 * ع

نق : نصف قطر القاعدة.   ع : الارتفاع

5- مستخدما مفاهيم البرمجة الشيئية ، اضف الى الفصيل السابق فصيل اخر اسمه UseCylinder والذي يحتوي على الدالة الرئيسية main() تقوم بقراءة قيمة نصف القطر القاعدة والارتفاع من لوحة المفاتيح ومن ثم تكوين كائن من فصيل الـCylinder وتمرير هذه القيم لبانيه ومن ثم طباعة حجم الاسطوانة على الشاشة.

التعابير الحسابية والمنطقية Expressions :

التعبير الحسابي مجموعة من المتغيرات والثوابت والدوال والمرتبطة بالعمليات الحسابية المعروفة (+ ، - ، * ، /) لإعطاء قيمة واحدة.

مثلاً :

```
Y = (X + 20) / 5;
```

تعبير حسابي

أما التعبير المنطقي مجموعة من العناصر المرتبطة بعلاقات منطقية وتكون نتيجتها اما (true) او (false) .

مثلاً :

```
Result = (X >= 40) && (Y != 20);
```

العمليات الحسابية Arithmetic Operations :

يمكن اجراء العمليات الحسابية على كلا من المتغيرات والاعداد الصحيحة والحقيقية ،حيث أن هناك عدد من المؤثرات التي يمكن استخدامها وهي :

+ علامة الجمع ، - علامة الطرح ، * علامة الضرب ، / علامة القسمة اضافة الى باقي القسمة % .

لاحظ انه اذا كان كلتا جانبي علامة القسمة (/) قيم صحيحة فالنتيجة ستكون صحيحة مع حذف الكسر مهما كانت قيمته ، أما اذا كانت احدى جانبي علامة القسمة عدد كسري فالنتيجة ستكون كسرية.

فمثلاً :

```
Y = 17 / 3;        ➔    5
Y = 17 / 3.0       ➔    3.66
```

اما باقي القسمة modulus الـ(%) فهي تنتج باقي القسمة من عددين صحيحين او كسريين :

```
Y = 17 Mod 5         ➔    2
Y = 17.5 Mod 5.0     ➔    2.5
```

حيث ان قسمة 17 على 5 هي 3 والباقي هو 2 .

وان قسمة 17.5 على 5.0 هو 3.0 والباقي 2.5 .

ملاحظة :

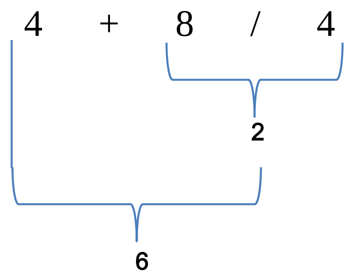
الاسبقية في تنفيذ العمليات الحسابية هي :

1- الاقواس اولا.

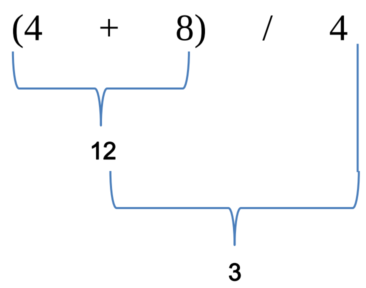
2- الضرب والقسمة من اليسار الى اليمين.

3- الجمع والطرح من اليسار الى اليمين.

فمثلاً :



أما



تلاحظ اختلاف الناتج عند استخدام الاقواس ، ففي الأولى تم اجراء عملية القسمة اولا والنتيجة هي 2 ثم تمت عملية الجمع مع الـ4 فالناتج النهائي كان 6.

اما الثانية فتمت اجراء ما بداخل الاقواس أولا والنتيجة كانت 12، ومن تم قسمت هذه النتيجة على 4 فكان الناتج هو 3.

العمليات المنطقية Logical Operations :

هي تعابير تحتوي على مجموعة من العناصر المرتبطة بمؤثرات علائقية ومنطقية ويكون ناتجها النهائي اما *true* أو *false* .

المؤثرات العلائقية هي :

== يساوي	!= لا يساوي
> أكبر من	< اصغر من
>= أكبر من أو يساوي	<= اصغر من او يساوي

المؤثرات المنطقية هي :

&& (و) : وتكون نتيجته صحيحة فقط في حال ما كان كلا من العنصرين صحيحين.

|| (أو) : وتكون نتيجته خاطئة فقط في حال ما كان كلا من العنصرين خاطئين.

! (نفي) : نفي العنصر.

!	النتيجة
`! True`	`False`
`! False`	`True`

||	النتيجة
`True  || True`	`True`
`True  || False`	`True`
`False || True`	`True`
`False || False`	`False`

&&	النتيجة
`True  && True`	`True`
`True  && False`	`False`
`False && True`	`False`
`False && False`	`False`

ان عملية التخصيص في الجافا تتم باستخدام علامة (=) حيث يكون يسارها اسم لمتغير ويمينها قيمة او نتيجة تعبير حسابي او نتيجة دالة ، فمثلا :

```
y = 5;
y = x + 10;
y = Math.pow(10, 2);
```

يمكن استخدام جملة التخصيص الممتد كالتالي :

```
y = x = z = 100;
```

حيث يتم تخصيص 100 لكل من z ثم x ثم y ، أي انها جميعا ستأخذ نفس القيمة 100.

ايضا يمكن استخدام التخصيص في اكثر من موضع في نفس التعبير ، فمثلا :

```
y = (x=10) + (z=20) ;
```

أي خصص 10 الى x ، ثم خصص 20 الى z ثم اجمع القيمتين (وهي هنا 30) وخصصها لـ y .

مؤثرات الزيادة ++ والنقصان -- :

حيث يمكن وضع مؤثر الزيادة ++ قبل او بعد المتغير والذي يؤدي الى زيادة قيمته بمقدار 1 وهناك فرق بين وضعه امام او بعد المتغير ، حيث ان وضع مؤثر الزيادة امام المتغير يعني اضف واحد اليه ثم قم بعملية التخصيص ، اما وضع مؤثر الزيادة ++ بعد المتغير يعني قم بعملية التخصيص اولا ثم اضف اليه واحد. فمثلا :

```
i = 10;
j = ++i;          /////      i = 11         j = 11

i = 10;
k = i++;          /////      i = 11         k = 10
```

وكذلك الحال بالنسبة الى مؤثر النقصان -- والذي يؤدي الى انقاص واحد من قيمة المتغير ويمكن وضعه امام او بعد المتغير حاله حال مؤثر الزيادة. قد يؤدي استخدام مؤثر الزيادة الى ارباك فمثلا :

```
i = 10;
i = ++i;
i = i++;
```

ماذا ستكون نتيجة الـ*i* في الاولى والثانية ؟

اذا كان استخدامهما سيؤدي الى عملية ارباك فيفضل استخدام الطريقة السهلة وهي :

```
i = i + 1;
```

المؤثرات المركبة :

يمكن استخدام المؤثرات الحسابية (+ ، - ، * ، / ، %) مع علامة التخصيص (=) كطريقة مختصرة للتخصيص كالتالي :

+= -= *= /= %=

فمثلا:

```
i = i  + 10;
```

يمكن كتابتها كالتالي :

```
i += 10;
```

ولكن !! كن حذرا عند استخدامك لهذا النوع من المؤثرات ، فمثلا ما الذي يناظره التعبير التالي :

y /= x+z; ???

هو يناظر التالي :

```
y = y/(x+z);
```

ولا يناظر التعبير التالي :

```
y = y/x+z;
```

الاختيار Selection :

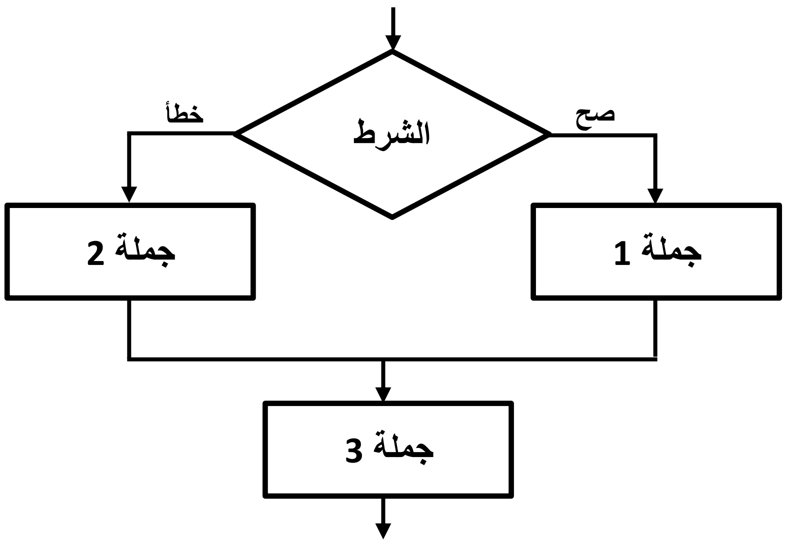
البرامج التي كتبناها حتى الان هي عبارة عن تسلسل من الجمل التي يتم تنفيذها سطر بعد الاخر، ولكن في البرامج الفعلية يفترض ان يسير البرنامج وفق اختيارات معينة وينفذ جمل بناء الى شروط معينة.

*جملة if

تقوم جملة if بتنفيذ جملة او عدة جملة اذا تحقق شرط ما والا فأنها تنفذ جمل اخرى بعد الـelse والشكل العام لها :

```
if ( شرط )
      جملــــــــــــــة 1;
else
      جملــــــــــــــة 2 ;

جملــــــــــــــة 3 ;
```



يعني اذا تحقق الشرط يتم تنفيذ جملة1 ، والا (أي اذا كان نتيجة الشرط خطأ) فسيتم تنفيذ الجملة2.

الجملة3 يتم تنفيذها بعد الخروج من جملة if في أيا من الحالتين.

جملة if يمكن ان تكون بالشكل التالي بدون جملة else :

```
if ( شرط )
     جملـــــــــــــة 1 ;
جملـــــــــــــة 3 ;
```

أي يتم تنفيذ جملة 1 اذا تحقق الشرط ، اما اذا كان الشرط خاطئ فينتقل التنفيذ الى الجملة3.

الشرط قد يكون بسيط او مركبا باستخدام (&& و || و !) فمثلاً :

```
if (avg >= 50) && (avg <= 100)
     System.out.println("النتيجة ناجح");
```

أي اذا كان avg اكبر من او يساوي 50 و اصغر من او يساوي 100 فسيطبع النتيجة ناجح.

مثال اخر :

```
if (income < 1000) || ( state.equals("exempting") )
      tax = 0;
else
      tax = income * 0.20;
System.out.println("Income : " + income +
                                     "\n Tax : "+ tax );
```

اذا كان الدخل income اصغر من 1000 <u>أو</u> (||) ان قيمة الحالة *state* هي معفي *"exempting"* فان الضريبة tax تحسب كـ0% والا (اي اذا كان الدخل اكبر من او يساوي 1000) فالضريبة تحسب كـ20%.

أما اذا اردنا تنفيذ اكثر من جملة واحدة في حالة تحقق الشرط فإننا نحصر كل هذه الجمل بين قوسي الفئة { } ليتم تنفيذها جميعا اذا تحقق الشرط او اهمالها جميعا اذا لم يتحقق الشرط كالتالي :

```
if !(income > 1000)
{
      tax = income * 0.05;
      net = income - tax;
}
else
{
      tax = income * 0.20;
      net = income - tax + (0.10 * income);
}
System.out.println("Net : " + net + "\n Tax : "+ tax );
```

هنا اذا كان الدخل ليس (!) اكبر من 1000 (أي اصغر من او يساوي 1000) فان الجملتين بين الـ{ } الاولى ستنفذ :

```
{
        tax = income * 0.05;
        net = income - tax;
}
```

حيث يتم حساب الضريبة كـ5% من الدخل ومن ثم حساب صافي الدخل بعد خصم الضريبة منه.

أما اذا كان الدخل اكبر من 1000 فان الجمل بين الـ{ } الثانية ستنفذ :

```
{
        tax = income * 0.20;
        net = income - tax + (0.10 * income);
}
```

حيث تحسب الضريبة كـ20% من الدخل ، ويحسب صافي الدخل بعد خصم الضريبة من اجمالي الدخل واضافة 10% من قيمة الاجمالي.

ثم بعد ذلك سيتم تنفيذ جملة طباعة صافي الدخل والضريبة بجملة الطباعة.

جملة if قد متداخلة كالتالي :

```
if ( شرط1 )
        جملـــــــــــــة 1;
else
        if ( شرط2 )
            جملـــــــــــــة 2;
        else
            If (شرط3 )
                جملـــــــــــــة 3;
            else
                جملـــــــــــــة 4 ;

جملـــــــــــــة 5 ;
```

أي اذا كان (الشرط1) صحيح فنفذ (الجملة1) ، والا (أي اذا كان الشرط1 خطأ) فنفذ الجملة التي بعد else وهي جملة if اخرى ، فاذا كان (الشرط2) صحيح فنفذ (الجملة2) والا نفذ جملة if الثالثة ... وهكذا الى ان يتم تنفيذ احدى الجمل، بعد ذلك نفذ (الجملة5) والتي هي خارج جملة if.

فمثلاً :

```
if (income < 1000)
    tax = income * 0.05;
else
    if (income < 5000)
        tax = income * 0.10;
    else
        tax = income * 0.20;
System.out.println("Tax : " + tax );
```

هنا اذا كان الدخل اقل من 1000 فالضريبة هي 5 %. والا اذا كانت الضريبة اقل من 5000 (طبعا اعلى من او يساوي 1000) فالضريبة هي 10%. والا (أي اذا كان الدخل اعلى من او يساوي 5000) فالضريبة هي 20%. ومن ثم طباعة الضريبة .

مثال :

اكتب برنامج يتكون من فصيل واحد يحتوي على الدالة الرئيسية main() و تقوم بقراءة قيمة من الشاشة تمثل دخل محل تجاري ومن ثم حساب نسبة الضريب وصافي الدخل حسب الاتي :

اذا كان الدخل اقل من 2000 فالضريبة 5%.

اذا كان الدخل اعلى من او يساوي 2000 واقل من 5000 فالضريبة 10%.

اذا كان الدخل 5000 فأعلى فالضريبة 15%.

```
import java.util.Scanner;
public class IfStatement
{
    public static void main(String[] args)
    {
        double income, tax, net;
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');
        System.out.print("Enter your Income : ");
        income = kb.nextDouble();
        if (income < 2000)
            tax = income * 0.05;
        else
            if (income < 5000)
                tax = income * 0.10;
            else
                tax = income * 0.15;
        net = income - tax;
        System.out.format(" Income : %.2f" , income );
        System.out.format("\n Tax : %.2f" , tax );
        System.out.format("\n net : %.2f" , net );
    }
}
```

مثال :

اكتب برنامج جافا يتكون من فصيلين الاول اسمه الشخص **Person** والذي يحتوي على حقلين هما الاسم ***name*** والعمر ***age*** ، ويحتوي على باني واحد يستقبل قيمتين للاسم والعمر . يحتوي هذا الفصيل ايضا على اربعة مناهج **الاول** يرجع قيمة الاسم **والثاني** يرجع قيمة العمر **والثالث** يزيد في العمر بمقدار واحد **والرابع** يحدد ويرجع حالة الشخص اما ان يكون قاصر *"minor"* (اقل من 18) او دافع للضريبة *"Tax-payer"* (بين 18 و 65) او متقاعد *"pensioner"* (اكبر من 65). اما الفصيل الثاني فاسمه **UsePerson** والذي يحتوي على الدالة الرئيسية *main()* و تقوم بقراءة قيمتين من الشاشة الاول تمثل الاسم والثاني العمر ومن ثم تكوين كائن من فصيل الشخص وتمرير هذا القيم لبانيه ، ثم زيادة عمر الشخص بمقدار واحد ثلاث مرات ومن ثم طباعة حالته على الشاشة.

الفصيل الاول person :

```
public class Person
{

     private String name;
     private int age;

     public Person(String n, int a)
     {
          name = n;
          age = a;
     }
     public String getName()
     {
         return name;
     }
     public int getAge()
     {
         return age;
     }
     public void increaseAge()
     {
         age++;
     }
     public String getStatus()
     {
         if (age < 18)
             return "Minor";
         else
              if (age >65)
                  return "Pensioner";
              else
                   return "Tax-payer";
     }

}
```

الفصيل الثانى Useperson :

```
import java.util.Scanner;

public class UsePerson
{
    public static void main(String[] args)
    {

        Scanner kb = new Scanner(System.in);

        System.out.print('\u000C');

        System.out.print("Enter Name : ");
        String n = kb.nextLine();

        System.out.print("Enter Age : ");
        int a = kb.nextInt();

        Person p1 = new Person(n, a);

        p1.increaseAge();
        p1.increaseAge();
        p1.increaseAge();

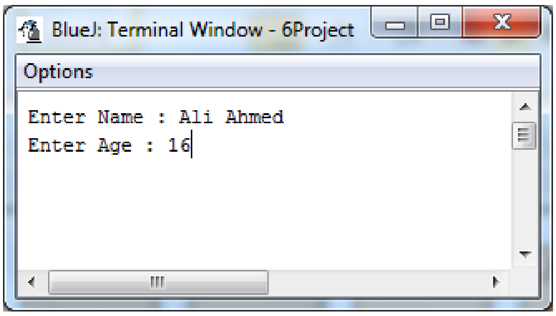
        System.out.println(" Name : " + p1.getName() );
        System.out.println(" Age now : " + p1.getAge() );
        System.out.println(" Status : " + p1.getStatus() );
    }
}
```

قراءة الاسم والعمر من الشاشة

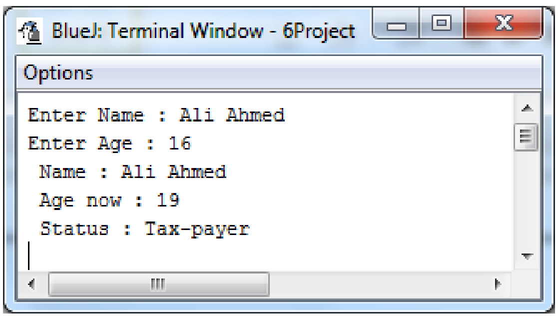
تكوين كائن من Person وارسال الاسم والعمر له

زيادة العمر 3 مرات بمقدار واحد في كل مرة

طباعة الاسم والعمر وحالة الشخص



بعد ادخال القيم



بعد زيادة العمر 3 مرات (بمقدار واحد في كل مرة) وطباعة حالة الشخص.

*جملة التبديل switch

تقوم جملة switch بتنفيذ واحد من الاختيارات المتاحة بناء على قيمة المتغير الرقمي او الرمزي كالتالي :

```
switch   ( متغير صحيح/رمزي )
{
    case  قيمة1     :     جملة1 ; break;
    case  قيمة2     :     جملة2 ; break;
                                .
                                .
    case  قيمة ن    :     جملة ن ; break;
default
              جملة م;
}
```

تقوم جملة التبديل switch باختبار المتغير الصحيح او الرمزي فاذا ساوى (قيمة1) فسيتم تنفيذ (الجملة1)، والا اذا ساوي (القيمة2) فسيتم تنفيذ (الجملة2) ... وهكذا ، اي سيتم تنفيذ احدى الجمل فقط ، وفي حالة عدم تساويها مع اي من القيم فسيتم تنفيذ (الجملة م) بعد الـdefault .

فمثلاً :

```
switch   (a)
{
     case 0  :    System.out.println("Zero");    break;
     case 1  :    System.out.println("One");     break;
     case 2  :    System.out.println("Two");     break;
     case 3  :    System.out.println("Three");   break;
     case 4  :    System.out.println("Four");    break;
     default  :   System.out.println("A is ERROR");
}
```

أي تختبر قيمة المتغير الصحيح a فاذا ساوي 0 فسيتم طباعة Zero والا اذا ساوى 1 فسيتم طباعة One ... وهكذا أما كان a لا يساوي أيا من 0,1,2,3,4 فسيتم طباعة 'A is ERROR'.

مثال اخر :

```
switch  (a)
{
  case 0:  case 1:
  case 7:  System.out.println("Small");  break;
  case 10: case 20: System.out.println("Middle"); break;
  case 70: System.out.println("Big");
}
```

لاحظ انه يمكن كتابة عدة قيم وراء بعضها كما لاحظنا (case 0 :   case 1:   case 7:)
اي اذا ساوى a أيا من هذه القيم فسيتم تنفيذ الجملة الاولى وطباعة "'Small'" ثم يصل الى break ليخرج خارج switch .

لاحظ ايضا انه عند الوصول الى break يتم قطع التنفيذ والخروج خارج جملة switch ، بمعني اذا نسيت جملة الـbreak سيتم تنفيذ الجملة التالية لها حتى يصل الى جملة break .

فمثلا :

```
switch  (a)
{
  case 0:  case 1:
  case 7:  System.out.println("Small");
  case 10: case 20: System.out.println("Middle"); break;
  case 70: System.out.println("Big");
}
```

لاحظ اننا لم نضع جملة ***break*** بعد جملة الطباعة الاولى ، فاذا كانت a تساوي 1 فسيتم طباعة :

```
Small
Middle
```

لان بعد جملة طباعة *Small* لا يوجد ***break*** لذلك استمر البرنامج بتنفيذ الحالة التالية وهي طباعة *Middle* .

ايضا جملة ***default*** اختيارية يمكن كتابتها او حذفها كما في المثال الاخير.

مثال اخر :

```
switch  (ch)
{
   case 'A':  case 'a' :   System.out.println("A");  break;
   case 'B':  case 'b' :   System.out.println("B");  break;
   case 'Z':  case 'z' :   System.out.println("Z");  break;
   case '1':  case '2' :   System.out.println("no"); break;

  default: System.out.println("Maybe a special character");
}
```

هنا يتم اختبار المتغير الرمزي فاذا كان 'A' أو 'a' فسيتم طباعة "A" ، اما اذا كان 'B' أو 'b' فسيتم طباعة "B" ... وهكذا ، اما اذا لم يساوى أيا من الرموز المعروضة ضم الـ case فسيتم طباعة " Maybe a special character " .

مثال :

اكتب برنامج يتكون من فصيل واحد يحتوي على الدالة الرئيسية ()main و تقوم بقراءة قيمتين صحيحتين تمثل الاول الشهر والثانية السنة ومن ثم طباعة عدد ايام الشهر في تلك السنة (هل هي 30 او 31 او 29 ...الخ).

حيث ان : هناك 31 يوم في الاشهر (1 ، 3 ، 5 ، 7 ، 8 ، 10 ، 12).

وهناك 30 يوم في الاشهر (4 ، 6 ، 9 ، 11).

وهناك 28 أو 29 يوم في شهر 2 اعتماد على السنة (هل هي بسيطة او كبيسة).

```
import java.util.Scanner;

public class DaysInMonth
{

 public static void main(String[] args)
 {

    Scanner kb = new Scanner(System.in);
    System.out.print('\u000C');

    System.out.print("Enter Month & Year : ");

    int month = kb.nextInt();
    int year = kb.nextInt();
    int num =0;

    switch (month)
    {
        case 4:    case 6:
        case 9:    case 11:  num = 30;  break;

        case 1: case 3: case 5:
        case 7: case 8: case 10: case 12: num= 31; break;

        case 2:  num = (year % 4 ==0)? 29:28; break;
        default : {
                      System.out.print("the input is error ");
                      System.exit(0);
                  }

      }
      System.out.println("Days in month("+month+") : " +
                                     num + " Days.");
 }
}
```

هنا قمنا بإدخال رقمين للشهر month والسنة year من لوحة المفاتيح ، ثم اختبرانا قيمة الشهر ضمن جملة التبديل switch :

في حالة ما اذا كان الشهر مساويا لأي من القيم (4 او 6 او 9 او 11) فان عدد الايام num تساوي 30 يوما.

وفي حالة ما اذا كان الشهر مساويا لأي من (1 او 3 او 5 او 7 أو 8 أو 10 أو 12) فان عدد الايام num تساوي 31 يوما.

اما في حالة شهر 2 فاننا استخدمنا المعامل الشرطي (?) والذي يستخدم للمقارنة بين قيمتين ويأخذ الشكل التالي :

تعبير منطقي1 ? تعبير2 : تعبير3 ;

اذا كانت قيمة التعبير الاول صحيحة true فانه تحسب قسمة التعبير الثاني ، والا اذا كانت قيمة التعبير الاول خطا false فانه يحسب قسمة التعبير الثالث.

فمثلا :

```
int a, b, c;
a = 10;
b = 20;
c =   (a<b) ? a  :  b;
```

هنا خصصنا 10 للـa و 20 للـb وقيمة الـc تعتمد على الاختبار فيما اذا كان (a<b) فان قيمة الـc تساوي a أي 10 ، اما اذا كانت a ليست اصغر من b فان الـc تساوي قيمة الـ b.

أي : اذا الشرط (a<b) صحيح فان c=a

اما اذا كان الشرط (a<b) خطا فان c= b

لنرجع لمثالنا السابق حيث كتبنا الجملة التالية عند ادخال شهر 2 :

```
num = (year % 4 ==0)? 29:28;
```

حيث انه اذا كانت السنة كبيسة (أي تقبل القسمة على 4 بدون باقي) فان عدد ايام شهر 2 (num) هي 29 يوم.

اما اذا كانت السنة بسيطة (أي لا تقبل القسمة على 4) فان عدد ايام شهر 2 (num) هي 28 يوم.

اما في حالة ادخال أي عدد للشهر (خارج المجال من 1 الى 12) فان الـ ***default*** ستنفذ وهي طباعة رسالة تفيد بان الشهر خطأ ويتم الخروج من البرنامج نهائيا باستخدام الجملة **exit(0)** والموجودة ضمن الفصيل ***System*** :

```
System.exit(0);
```

الحلقات أو التكرار Iteration

الى حد الان كنا نكتب برامج تنفذ لمرة واحدة فقط ، ولكن اغلب البرامج الفعلية تتطلب تنفيذ عدة جمل مرات عدة و هذا ما نسميه بالتكرار.

توفر لغة الجافا ثلاث انواع ن الحلقات وهي : (while ، for ، do…while)

* جملة طالما while :

الشكل العام لها :

```
while  ( شرط )
{
        جملة 1   ;
        جملة 1   ;
          .
          .
        جملة ن  ;
}
جمـــــــــــلة م  ;
```

فطالما كان الشرط صحيحا يتم تنفيذ الجملة من 1 الى ن ، وعندما يصبح الشرط خطأ يتم الخروج من الحلقة الى الجملة التالية لجملة while وهي (جملة م) .

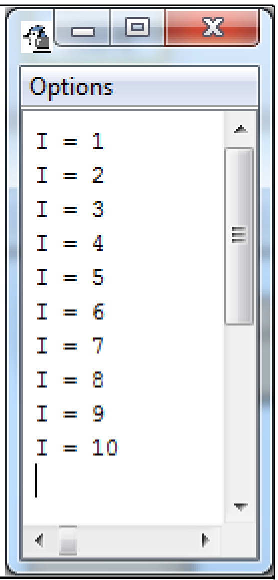
مثال :

اكتب برنامج لطباعة الاعداد من 1 الى 10 باستخدام جملة while .

```java
public class WhileLoop
{

    public static void main(String[] args)
    {
        System.out.print('\u000C');

        int i = 1;
        while ( i<=10 )
        {
            System.out.println("I = " + i);
            i++;
        }
     }
}
```



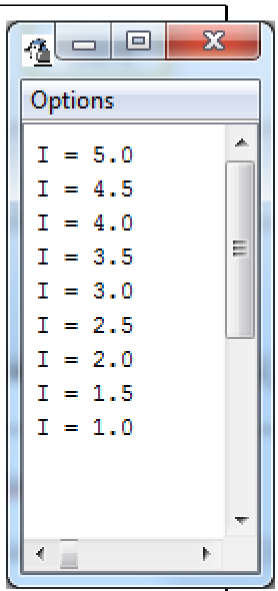
لاحظ انه تم وضع قيمة ابتدائية للعداد i وهي 1 قبل الدخول في الحلقة ، بعدها تم اختبار الشرط وهو (i <= 10) اي طالما العداد i اصغر من او يساوي 10 نفذ الجمل الواقعة بين { } ، لاحظ ايضا ان قبل نهاية الحلقة تم زيادة قيمة العداد i بواحد ثم يتم الرجوع مجددا لاختبار قيمة العداد ، وهكذا تستمر الحلقات في التنفيذ الى ان تصبح قيمة العداد i اكبر من 10 فيتم الخروج من الحلقة.

مثال :

اكتب برنامج لطباعة الاعداد تنازليا من 5 الى 1 وبمقدار نقصان 0.5 في كل مرة.

```
public class DescendingWhile
{
    public static void main(String[] args)
    {
        System.out.print('\u000C');

        double i = 5.0;
        while (i >= 1.0)
        {
              System.out.println("I = " + i);
              i-= 0.5;     ///  OR  i = i - 0.05
        }
    }
}
```



لاحظ ان :

* القيمة الابتدائية للعداد حقيقية هي 5.0 وان الشرط هو (i>=1.0) ويتم انقاص 0.5 في كل مرة.

* تغيير (زيادة او انقاص) قيمة العداد يجب ان تكون داخل الحلقة (اي داخل { }) ، والا فسوف يتم تنفيذ الحلقة الى ما لانهاية.

يمكن ان تستخدم while بشكل متداخل بحيث يتم تنفيذ حلقتين او اكثر واحدة داخل الاخرى كما بالمثال التالي :

مثال :

اكتب برنامج لطباعة جدولي ضرب 7 و 8 :

```
public class NestedWhile
{

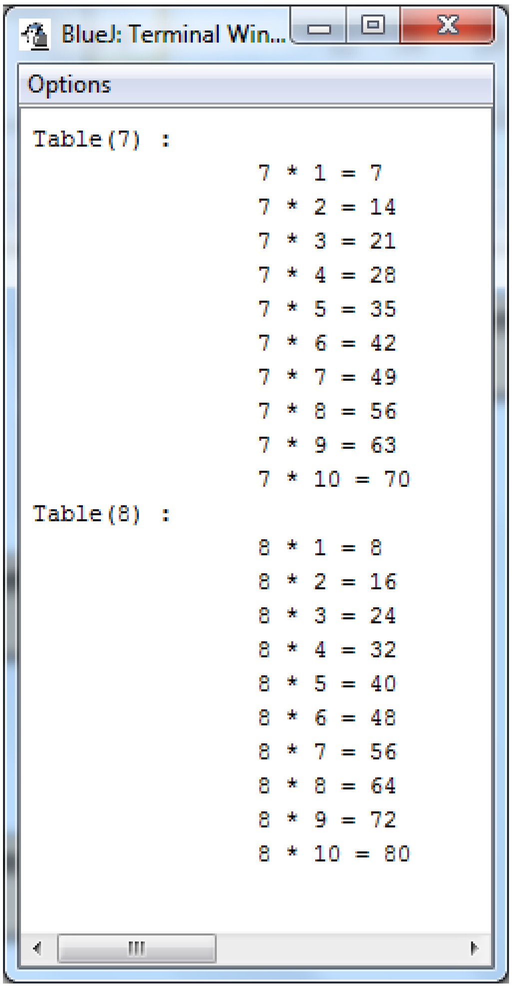
  public static void main(String[] args)
  {
     System.out.print('\u000C');

     int i = 7;
     while (i<=8)
     {

         System.out.println("Table(" + i + ") : ");

         int k = 1;
         while (k <= 10)
         {
            System.out.println("\t\t"+i + "*" + k + "="+ i*k);
            k++;
         }

         i++;
     }
  }
}
```



* جملة كرر ... طالما do...while :

الشكل العام لها :

```
do
        جملة 1 ;
        جملة 1 ;
        .
        .
        جملة ن ;
while (شرط) ;

جمـــــــــــــلة م ;
```

اي كرر الجمل الواقعة بين do ... while طالما أن الشرط صحيح ، فعندما يصبح الشرط خطأ يتم الخروج من الحلقة وينفذ (الجملة م).

مثال :

اكتب برنامج لطباعة الاعداد من 1 الى 10 باستخدام جملة do...while .

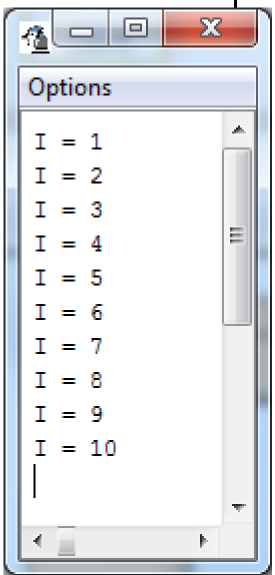
```
public class DoWlileLoop
{
    public static void main(String[] args)
    {

        System.out.print('\u000C');

        int i = 1;

        do
        {
                System.out.println("I = " + i);
                i++;
        }  while (i<=10);

    }
}
```



لاحظ انه تم وضع قيمة ابتدائية للعداد i قبل الدخول للحلقة وهي 1 ثم الدخول في الحلقة وطباعة قيمة i ، بعدها تم زيادة قيمة العداد بواحد (i++) ، ثم اختبار الشرط وهي طالما i اصغر من 10 كرار الجمل الواقعة بين do...while الى ان يصبح الشرط خطأ بعدها يتم الخروج من الحلقة.

لاحظ أن :

* الجمل الواقعة بين do…while سيتم تنفيذها مرة واحدة على الاقل حتى وان كان الشرط خطأ اصلا.

* يتم وضع الجمل الواقعة في الحلقة بين قوسي الفئة {   } .

مثال :

اكتب برنامج لطباعة الاعداد الفردية ومجموع الاعداد الزوجية الواقعة بين 40 و 60 باستخدام جملة do…while .

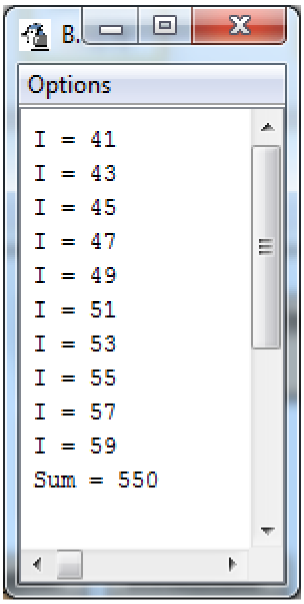
```
public class OddEvenNOs
{

    public static void main(String[] args)
    {
        System.out.print('\u000C');

        int i = 40,    sum = 0;

        do
        {
           if (i%2==1)
                System.out.println("I = " + i);
           else
                sum = sum + i;          //// sum += i;
           i++;
        }  while (i<=60);

        System.out.println("Sum = " + sum);
    }
}
```



<u>لاحظ أن</u> : لمعرفة ان العدد فر دي او لا ، قمنا باختبار باقي قسمة العدد على 2 فاذا كان باقي القسمة = 1 فالعدد فردي والا فالعدد زوجي.

* جملة لأجل for :

الشكل العام لها :

for (القيمة المبدئية ; الشرط ; مقدار الزيادة او النقصان)
{
جملة 1 ;
جملة 1 ;
.
جملة ن ;
}
جمـــــــــــلة م ;

أي كرر الجمل من (1 الى ن) ابتداء من القيمة المبدئية عدد من المرات وبمقدار الزيادة المحددة في كل مرة الى أن يصبح الشرط خطأ ثم اخرج من الحلقة ونفذ (الجملة م) .

فمثلا لطباعة الاعداد من 1 الى 10 نكتب الاتي :

```
for (int i = 1; i<=10 ; i++)
     System.out.println("I = " + i);
```

لاحظ انه اذا كانت جملة واحدة ضمن حلقة التكرار يمكن الاستغناء عن { } .

يمكننا كتابة القيمة المبدئية للعداد قبل جملة for مع ترك الفاصلة المنقوطة كالتالي :

```
int i = 1;
for (      ; i<=10 ; i++)
{
     System.out.println("I = " + i);
}
```

ايضا يمكننا كتابة مقدار الزيادة او النقصان (i++) ضمن جملة for كالتالي :

```
int i = 1;
for (      ; i<=10 ;     )
{
     System.out.println("I = " + i);
     i++;
 }
```

لاحظ ترك الفواصل المنقوطة كما هي وبدون حذف . الآن لاحظ ان جملة الـfor هي اشبه ما يكون الى جملة while. ايضا حلقة for قد تكون متداخلة كما ستلاحظها في الامثلة القادمة.

مثال :

اكتب برنامج لطباعة الاعداد ومجموعها والواقعة بين عددين مدخلين من لوحة المفاتيح .

```
import java.util.Scanner;

public class ForLoop
{

    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

        System.out.print("Enter 2 Values : ");
        int n1 = kb.nextInt();
        int n2 = kb.nextInt();

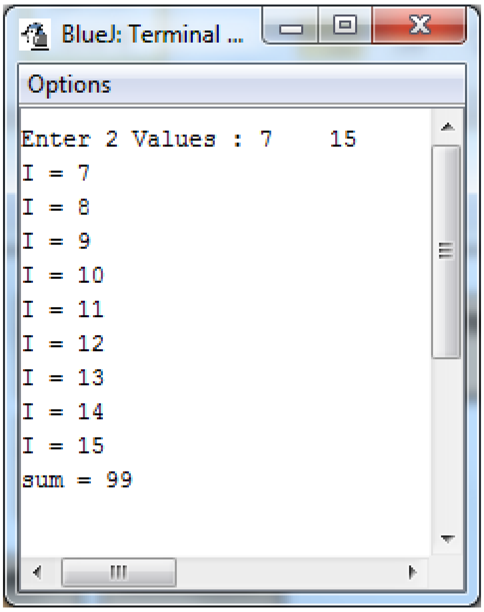
        int sum =0;

        for (int i=n1; i<=n2 ; i++)
        {
               System.out.println("I = " + i);

               sum = sum + i;    //// sum+=i;
         }

         System.out.println("sum = " + sum);

    }
}
```



مثال :

أكتب برنامج جافا مكون من فصيل اسمه ***Factorial*** يحتوي فقط الدالة الرئيسية ***main***() لإيجاد مضروب عدد مدخل من الشاشة.

حيث أن مضروب العدد ن = 1*2*3.......*ن

```
import java.util.Scanner;

public class Factorial
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

        System.out.print("Enter Integer Number : ");
        int n = kb.nextInt();

        long fact = 1;

        for (int i = 1; i<=n ; i++)
        {

              fact = fact * i;          //// fact *= i;
         }

         System.out.println(" Factorial = " + fact);
    }

}
```

لاحظ ان مضروب قيمة ***fact*** المبدئية هي 1 لأنه سيتم تخزين حاصل ضرب الاعداد بها ، ايضا تم استخدام ***long*** لأنه قد يكون المضروب (حاصل الضرب) كبير.

مثال :

أكتب برنامج لطباعة الشكل التالي لعدد من السطور N حيث ان N هو عدد مدخل من الشاشة يمثل عدد السطور :

```
Enter Integer Number : 5

*
***
*****
*******
*********
```

```
import java.util.Scanner;

public class Stars
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

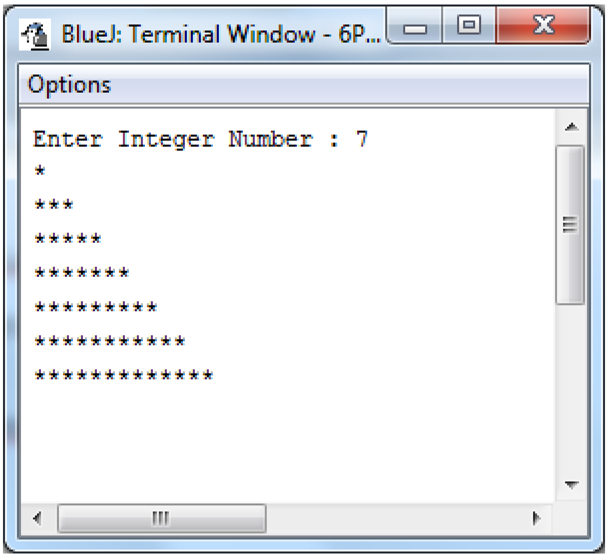
        System.out.print("Entern Integer Number : ");
        int n = kb.nextInt();


        for (int i = 1; i<=n ; i++)
        {
             for (int k = 1; k<=2*i-1; k++ )
                  System.out.print("*");

             System.out.print("\n");
         }
    }
}
```

لاحظ أن : * الحلقة الداخلية والتي بها العداد k تقوم بطباعة كل صف نجوم لوحده ، وان شرط النهاية هو (2*i-1) ، وبعد الانتهاء من الحلقة الداخلية يتم الانتقال الى سطر جديد باستخدام "\n".

* ان الحلقة الخارجية ستتكرر عدد من المرات بناء على قيمة n المدخلة من الشاشة.



مثال :

أكتب برنامج جافا مكون من فصيل اسمه ***Exercise*** يحتوي فقط الدالة الرئيسية ***main*()** لحساب المتوالية التالية :

$$y = x + x^2 + x^3 + \dots\dots + x^n$$

حيث N و X مدخلة من الشاشة.

```
import java.util.Scanner;

public class Exercise
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

        System.out.print("Enter Number X, N : ");
        double x = kb.nextInt();
        int    n = kb.nextInt();

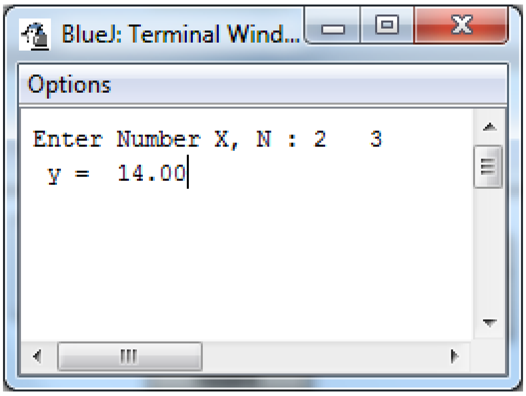
        double y = 0;

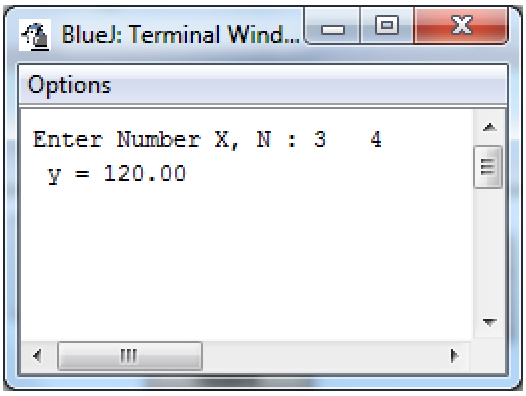
        for (int i = 1; i<=n ; i++)
        {

           y = y + Math.pow(x, i);    ////  y += Math.pow(x, i);
         }

         System.out.format(" y = %6.2f" , y);

    }
}
```





مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***InverseString*** يحتوي فقط الدالة الرئيسية ***main***() لإدخال سلسلة رمزية وطباعتها عكسيا.

```
import java.util.Scanner;

public class InverseString
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

        System.out.print("Enter String : ");
        String s = kb.nextLine();


        System.out.print(" The Inverse String : ");

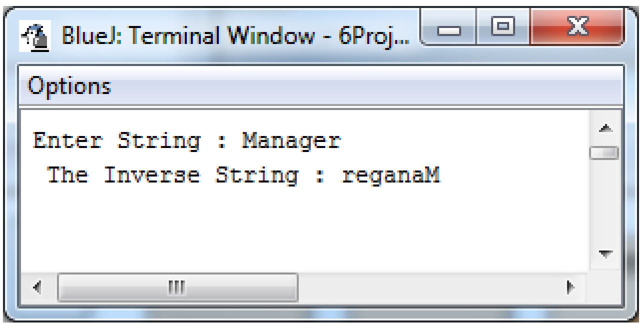
        for (int i = s.length()-1 ; i>=0 ; i--)
        {
               System.out.print( s.charAt(i) );
         }

    }
}
```

s

'M'	'a'	'n'	'a'	'g'	'e'	'r'
0	1	2	3	4	5	6

لاحظ أن الحلقة تبدا من (طول السلسلة -1) وهو رقم فهرس اخر رمز في المصفوفة ، وهي حلقة تنازلية الى ان تصل الى الصفر ، ثم تقوم بطباعة كل حرف لوحده باستخدام الدالة ***charAt(i)*** الموجودة ضمن الفصيل ***String*** .



جملة اقطع break :

سبق وأن شاهدنا هذه الجملة مع جملة التبديل switch والتي تستخدم لقطع تنفيذ بقية الحالات التالية لها ضمن جملة التبديل.

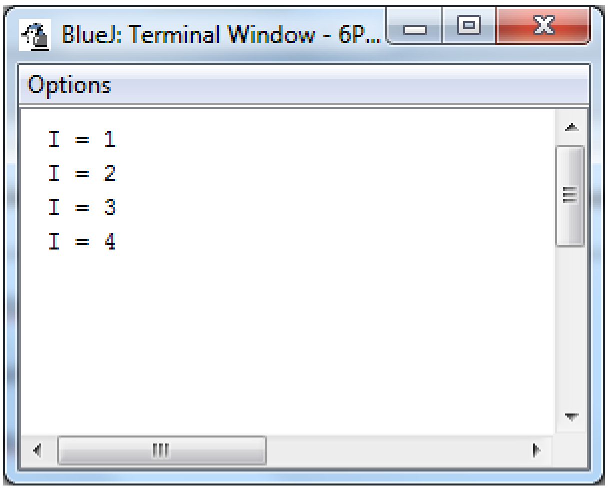
ايضا قد تستخدم هذه الجملة لقطع تنفيذ الحلقات مثل حلقة for والخروج خارج الحلقة فمثلا :

```
public class BreakStatement
{
    public static void main(String[] args)
    {
        System.out.print('\u000C');

        for (int i = 1; i<=10 ; i++)
        {
                if (i == 5 )
                    break;

                System.out.println(" I = " + i);
            }
    }
}
```

هنا الحلقة تبدا من 1 الى 10 ولكن ضمن جملة if يتم اختبار قيمة i فاذا ساوت 5 يتم تنفيذ جملة القطع break والخروج من الحلقة الى خارجها وبالتالي يتم طباعة الاعداد من 1 الى 4 فقط .



لاحظ انه اذا كانت جملة break ضمن الحلقة الداخلية يتم القطع والخروج فقط من الحلقة الداخلية ويستمر تنفيذ الحلقة الخارجية فمثلاً :

```
public class BreakStatement2
{
    public static void main(String[] args)
    {

        System.out.print('\u000C');

        for (int i = 1; i<=3 ; i++)
        {
            System.out.println(" I = " + i);

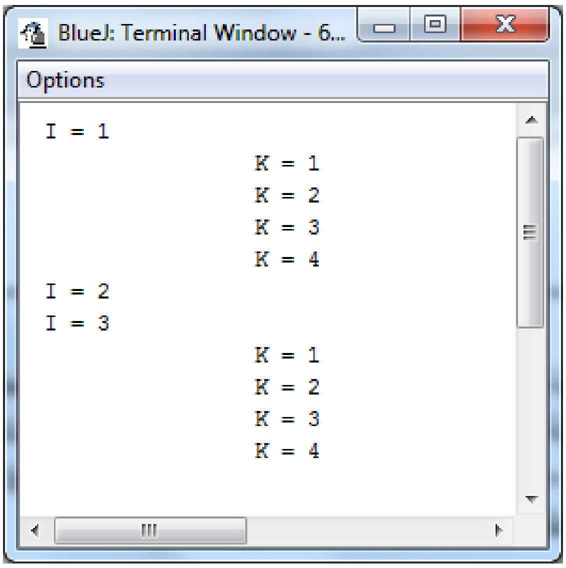
            for (int k = 1; k<=4 ; k++)
            {

                if (i == 2 )
                  break;

                System.out.println(" \t\t K = " + k);

            }
        }
    }
}
```

لاحظ ان الحلقة الخارجية i ستنفذ ثلاثة مرات ، وفي كل مرة سيتم تنفيذ الحلقة الداخلية k اربعة مرات ولكن لوجود جملة if بالحلقة الداخلية فسيتم قطع الحلقة الداخلية مباشرة والخروج خارج الحلقة الداخلية عندما I == 2 وبالتالي سوف لن يتم طباعة أيا من جمل الـk عندما الـi تساوي 2 ويستمر بتكملة تنفيذ الحلقة الخارجية والنتيجة ستكون كالتالي :



جملة استمر continue :

حالها حال جملة break في انها تستخدم للتحكم في سير تنفيذ البرنامج ، ولكنها لا تخرج من الحلقة ، بل تقوم بتوجيه التحكم الى نهاية الحلقة ومن ثم الاستمرار في بقية الحلقات .

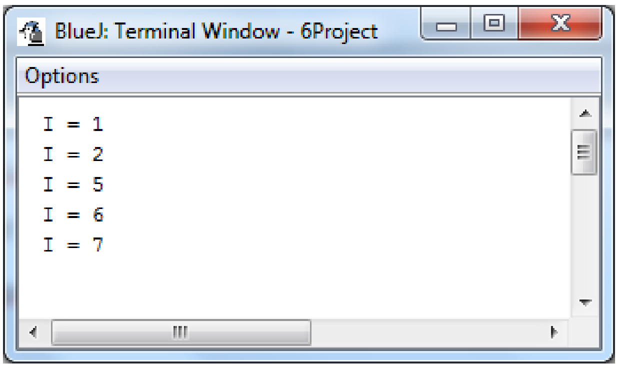
فمثلا :

```
public class ContinueStatement
{
    public static void main(String[] args)
    {
        System.out.print('\u000C');

        for (int i = 1; i<=7 ; i++)
        {
            if ( (i == 3 ) || (i == 4))
                 continue;
            System.out.println(" I = " + i);

        }
    }
}
```

هنا الحلقة for تبدأ من 1 الى 7 ولكن ضمن جملة الاختبار if يتم اختبار ما اذا كان قيمة i مساوية لـ 3 أو 4 بحيث سيتم توجيه التحكم الى نهاية حلقة for ولا يتم طباعة قيمة i في هذين الحالتين بل الاستمرار في بقية الحلقات ، فالنتيجة تكون كالتالي :



ايضا لا يخفى علينا وجود جملة ***exit(0)*** والتي تؤدي الى **انهاء تنفيذ البرنامج** كاملا وصيغتها كالتالي :

```
System.exit(0);
```

الوراثة Inheritance

من اقوى مفاهيم البرمجة الشيئية OOP هو مفهوم الوراثة ، بحيث تتيح امكانية تكوين فصيل بكامل محتوياته (حقول ومناهج) ومن ثم اعتبراها كأساس لفصائل اخرى ترث كل هذه الحقول والفصائل مع امكانية اضافة حقول ومناهج اخرى خاصة بها ، يعرف الفصيل الاولى بـ(superclass) او الفصيلة الاساسية (basic class) اما الفصائل الفرعية (subclass) فتعرف بالفصائل المشتقة (derived class) . الوراثة ستتيح لنا الاستغناء عن اعادة نفس البرمجة عدة مرات اضافة الى انه يمكن اعادة برمجة احدى المناهج الموجودة في الفصائل الاساسية ضمن احدى الفصائل الفرعية وهو ما يعرف بتجاوز المناهج (overriding) والتي تمثل احدى صور تعدد الاشكال (Polymorphism) ضمن البرمجة الشيئية .

<u>**فمثلا**</u> : ان الكادر البشري في الجامعة (University Persons) هم بالأصل اشخاص (Persons) وهم ينقسمون اما طلبة (Students) او مدرسين (Teachers) .

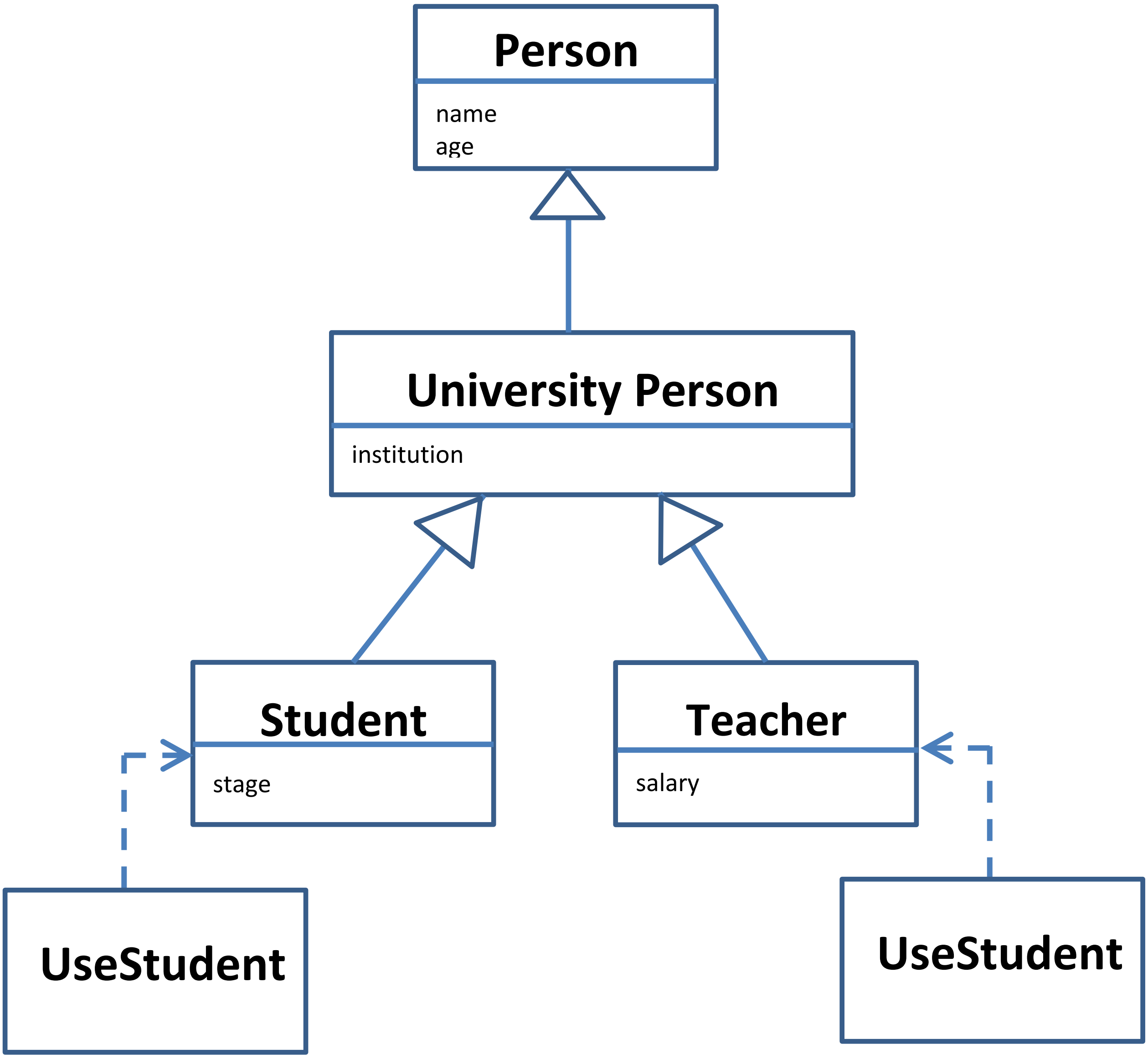
ولكونهم اشخاص (Person) فلهم خصائص مشتركة مثل الاسم ***name*** والعمر ***age*** .

ولكن الكادر البشري في الجامعة له خصائص وتصرفات تختلف عن بقية البشر الاخرين فمثلا له صفة الانتماء الى الجامعة ، فهو ينتمي الى مؤسسة (***institution***) الجامعة .

ايضا للطالب (Student) خصائص تميزه عن المدرس (Teacher) . فمثلا الطالب يدرس بمرحلة (***stage***) معينة بعكس المدرس الذي يتقاضى مرتب (***salary***) . فميزة **المرحلة** مثلا ينفرد بها الطالب ، اما ميزة **المرتب** فينفرد بها المدرس مع ان كلاهما بشر(Person) وينتمون الى نفس المؤسسة (University Person).

يمكن توضيح الوراثة بالمخطط التالي ، حيث نلاحظ ان الاسم (name) والعمر(age) موجود ضمن الفصيل الاساسي (Person) ، وان فصيل شخص الجامعة (UniversityPerson) يرث هذين الخاصيتين ويضف اليها خاصية اخرى هي صفة المؤسسة (institution) التي ينتمي اليها الشخص ، اما فصيل الطالب (Student) فيرث خاصيتي الاسم والعمر (name, age) وكذلك خاصية المؤسسة (institution) من فصيل شخص الجامعة (UniversityPerson) ويضيف اليها خاصية المرحلة (stage) . وبالنسبة الى فصيل المدرس (Teacher) فيرث ايضا خاصيتي الاسم والعمر (name, age) وكذلك خاصية المؤسسة (institution) من فصيل شخص الجامعة (UniversityPerson) ويضيف اليها خاصية المرتب (salary).

وبهذا يمكننا تكوين فصائل لاستخدام كل من فصيل الطالب (Student) وليكن (UseStudent) وايضا لاستخدام فصيل المدرس (Teacher) وليكن (UseTeacher).



من الشكل السابق نلاحظ استخدامنا للاسهم المتواصلة لتعني الفصيل الفرعي يرث من الاساسي ، وكذلك الاسهم المتقطعة لتعني ان فصيل ما يستخدم فصيل اخر (أي يكون كائن object منه بداخله) .

كل الفصائل التي نكونها في الجافا هي بالأساس فصائل فرعية من الفصيل الاساس **Object** حيث ترث بعض المناهج بشكل مجاني مثل :

```
toString()
equals()
Notify()
HashCode()
Wait()
```

والتي يمكننا اعادة برمجتها وفق ما نريد (*Overriding*)

➢ الان سنقوم ببرمجة اول فصيل وهو Person :

```
public class Person
{

    private String name;
    private int age;

    public Person(String pName, int pAge)
    {
        name = pName;
        age  = pAge;
    }


    public String getName()
    {
        return name;
    }


    public int getAge()
    {
        return age;
    }


    public void printDetails()
    {
         System.out.println( "Name : " + getName() );
         System.out.println( "Age : " + age );

    }

}
```

قمنا بتعريف حقلين خاصين هما الاسم name والعمر age مع تعيين قيم لها ضمن الباني وكتابة ثلاثة مناهج اثنان لترجيع هذه القيم وواحد لطباعة هذين الحقلين ***printDetails()*** .

➢ اما فصيل شخص الجامعة هو UniversityPerson :

```
public class UniversityPerson extends Person
{
    private String institution;

    public UniversityPerson(String n, int a, String inst)
    {
       super (n, a);
       institution = inst;
    }

    public String getInstitution()
    {
        return institution;
    }

    public void printDetails()
    {
         super.printDetails();
         System.out.println( "Institution : " + institution );
    }
}
```

في هذا الفصيل قمنا بتعريف فقط حقل خاص واحد هو حقل المؤسسة ***institution*** . وان حقل الاسم والعمر سوف يتم وارثته من الفصيل ***Person*** فلا داعي لإعادة كتابته من جديد .

تمت الوراثة بكتابة جملة ***extends Person*** في اول سطر بعد اسم الفصيل والذي يعني ان هذا الفصيل هو امتداد لفصيل الشخص ***Person*** .

اول جملة ضمن الباني هي استدعاء لباني الـPerson عن طريق كتابة كلمة **super** ثم تمرير قيمتي الاسم (***n***) والعمر (***a***) اليه ليتم اسنادها الى حقلي الاسم والعمر في فصيل الـ Person.

ايضا ضمن الباني يتم تخصيص قيم لباقي حقول الفصيل وهو هنا المؤسسة ***institution***.

لاحظ اننا قمنا بطباعة تفاصيل شخص الجامعة بنفس اسم المنهج الذي استخدمناه ضمن فصيل الـPerson وهو ***printDetails()*** وهذا ما يسمى بتجاوز المنهج Override ، أي استخدمنا نفس اسم منهج موجود بالفصيل الاساس وبنفس نوع وعدد البارامترات .

لاحظ ايضا اننا عند استدعائنا لأي منهج ضمن الفصيل الاساس (***superclass***) فإننا نكتب كلمة **super** ثم نقطة (**.**) ثم اسم المنهج المراد استدعائه وهو هنا ***printDetails()*** كالتالي :

```
super.printDetails();
```

والذي يقوم بطباعة الاسم والعمر ومن ثم اضافنا عليه ضمن منهج طباعتنا الجديد جملة طباعة المؤسسة.

```
System.out.println( "Institution : " + institution );
```

➢ اما فصيل الطالب هو Student :

```
public class Student extends UniversityPerson
{

    private int stage;

    public Student(String n, int a, String inst, int st)
    {
        super(n, a, inst);
        stage = st;
    }

    public int getStage()
    {
       return stage;
    }

    public void printDetails()
    {
         super.printDetails();
         System.out.println( "Stage : " + stage );

    }
}
```

في هذا الفصيل قمنا بتعريف فقط حقل خاص واحد هو حقل المرحلة ***stage*** . وان حقل الاسم والعمر والمؤسسة سوف يتم وارثته من الفصيل ***UniversityPerson*** فلا داعي لإعادة كتابته من جديد .

تمت الوراثة بكتابة جملة ***extends Person*** في اول سطر بعد اسم الفصيل والذي يعني ان هذا الفصيل هو امتداد لفصيل شخص الجامعة ***UniversityPerson***.

اول جملة ضمن الباني هي استدعاء لباني الـ***UniversityPerson*** عن طريق كتابة كلمة **super** ثم تمرير قيم الاسم (***n***) والعمر (***a***) والمؤسسة (***inst***) اليه ليتم اسنادها الى حقول الاسم والعمر والمؤسسة في فصيل الـ ***UniversityPerson***.

ايضا ضمن الباني يتم تخصيص قيم لباقي حقول الفصيل وهي هنا المرحلة ***stage***.

لاحظ اننا قمنا بطباعة تفاصيل الطالب ***student*** بنفس اسم المنهج الذي استخدمناه ضمن فصيل الـ***Person*** وفصيل الـ***UniversityPerson*** وهو ***printDetails()*** .

لاحظ ايضا اننا قمنا باستدعاء منهج الطباعة الخاص بـ(***UniversityPerson***) بكتابة كلمة **super** ثم نقطة (**.**) ثم اسم المنهج ***printDetails()*** ، اضافة جملة طباعة المرحلة *stage* .

➢ اما فصيل المدرس هو Teacher :

```
public class Teacher extends UniversityPerson
{
    private double salary;

    public Teacher(String n, int a, String inst, double sal)
    {
        super(n, a, inst);
        salary = sal;
    }

    public double getSalary()
    {
       return salary;
    }

    public void printDetails()
    {
         super.printDetails();
         System.out.println( "Salary : " + salary );

    }
}
```

في هذا الفصيل قمنا بتعريف فقط حقل خاص واحد هو حقل المرتب ***salary*** . وان حقل الاسم والعمر والمؤسسة سوف يتم وارثته من الفصيل ***UniversityPerson*** فلا داعي لإعادة كتابته من جديد .

تمت الوراثة بكتابة جملة ***extends Person*** في اول سطر بعد اسم الفصيل والذي يعني ان هذا الفصيل هو امتداد لفصيل شخص الجامعة ***UniversityPerson***.

اول جملة ضمن الباني هي استدعاء لباني الـ***UniversityPerson*** عن طريق كتابة كلمة **super** ثم تمرير قيم الاسم (***n***) والعمر (***a***) والمؤسسة (***inst***) اليه ليتم اسنادها الى حقول الاسم والعمر والمؤسسة في فصيل الـ ***UniversityPerson***.

ايضا ضمن الباني يتم تخصيص قيم لباقي حقول الفصيل وهي هنا المرتب ***salary***.

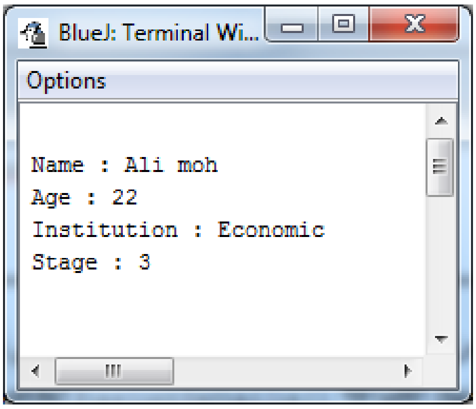
لاحظ اننا قمنا بطباعة تفاصيل المدرس ***Teacher*** بنفس اسم المنهج الذي استخدمناه ضمن فصيل الـ***Person*** وفصيل الـ***UniversityPerson*** وهو ***printDetails()*** .

لاحظ ايضا اننا قمنا باستدعاء منهج الطباعة الخاص بـ(***UniversityPerson***) بكتابة كلمة **super** ثم نقطة (**.**) ثم اسم المنهج ***printDetails()*** ، اضافة جملة طباعة المرتب *salary* .

➢ الان سنقوم بكتابة فصيل (***UseStudent***) لاستخدام فصيل الطالب ***Student*** :

```
public class UseStudent
{
    public static void main(String[] args)
    {
      Student ali = new Student("Ali moh", 22, "Economic", 3);
      System.out.println('\f');
      ali.printDetails();
    }
}
```

لاحظ اننا كونا كائن اسمه ***ali*** من فصيل الطالب ***Student*** وارسالنا الى بانيه القيم الموضحة اعلاه ، ومن ثم يمكننا استدعاء منهج الطباعة `printDetails()` والخاص بالطالب ***ali*** حيث سيقوم بطباعة اسمه وعمره ومؤسسته ومرحلته كالتالي :



لاحظ ان طباعة الاسم والعمر تتم في فصيل الشخص ***Person*** ، وان طباعة المؤسسة تتم في فصيل شخص الجامعة ***UniversityPerson*** ، وان المرحلة تتم طباعتها في فصيل الطالب ***Student*** والذي يقوم باستدعاء مناهج الطباعة في الفصائل الاعلى مرتبة.

بدلا من استدعاء منهج الطباعة

```
ali.printDetails();
```

يمكن كتابة الاتي جمل الطباعة التالية لطباعة نفس النتيجة السابقة :

```
System.out.println( ali.getName() );
System.out.println( ali.getAge() );
System.out.println( ali.getInstitution() );
System.out.println( ali.getStage() );
```

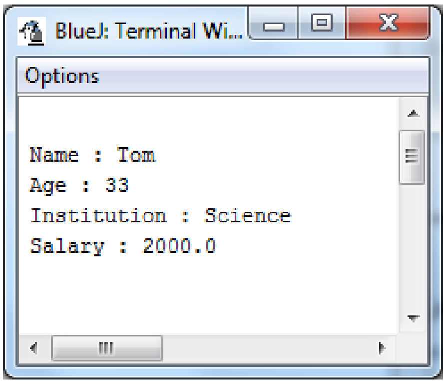
ولكن ***ali*** طالب ***Student*** فمن أين جاء منهج ***getName()*** ونحن لم نبرمجه في هذا الفصيل ؟ والاجابة لأنه ورثه من الفصيل الاعلى منه ***UniversityPerson*** والذي ورثه هو بدوره من الفصيل الاعلى وهو ***Person*** .

➢ الان سنقوم بكتابة فصيل (***UseTeacher***) لاستخدام فصيل الطالب ***Teacher*** :

```
public class UseTeacher
{
    public static void main(String[] args)
    {
       Teacher tom = new Teacher("Tom", 33, "Science", 2000);
       System.out.println('\f');

       tom.printDetails();
    }
}
```

لاحظ اننا كونا كائن اسمه ***tom*** من فصيل المدرس ***Teacher*** وارسالنا الى بانيه القيم الموضحة اعلاه ، ومن ثم يمكننا استدعاء منهج الطباعة `printDetails()` والخاص بالمدرس ***tom*** حيث سيقوم بطباعة اسمه وعمره ومؤسسته ومرتبه كالتالي :



لاحظ اننا يمكننا تكوين فصيل يستخدم كل من فصيل الطالب والمدرس معا :

```
public class UseTeacherStudent
{
    public static void main(String[] args)
    {
       Teacher tom = new Teacher("Tom", 33, "Science", 2000);
       Student ali = new Student("Ali moh",22,"Economic", 3);

       System.out.println('\f');

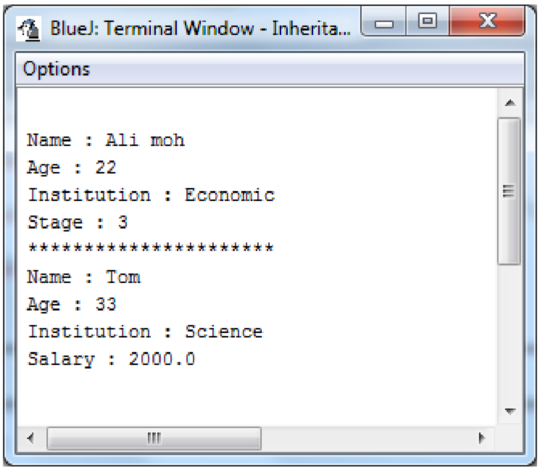
       ali.printDetails();

       System.out.println("**********************");

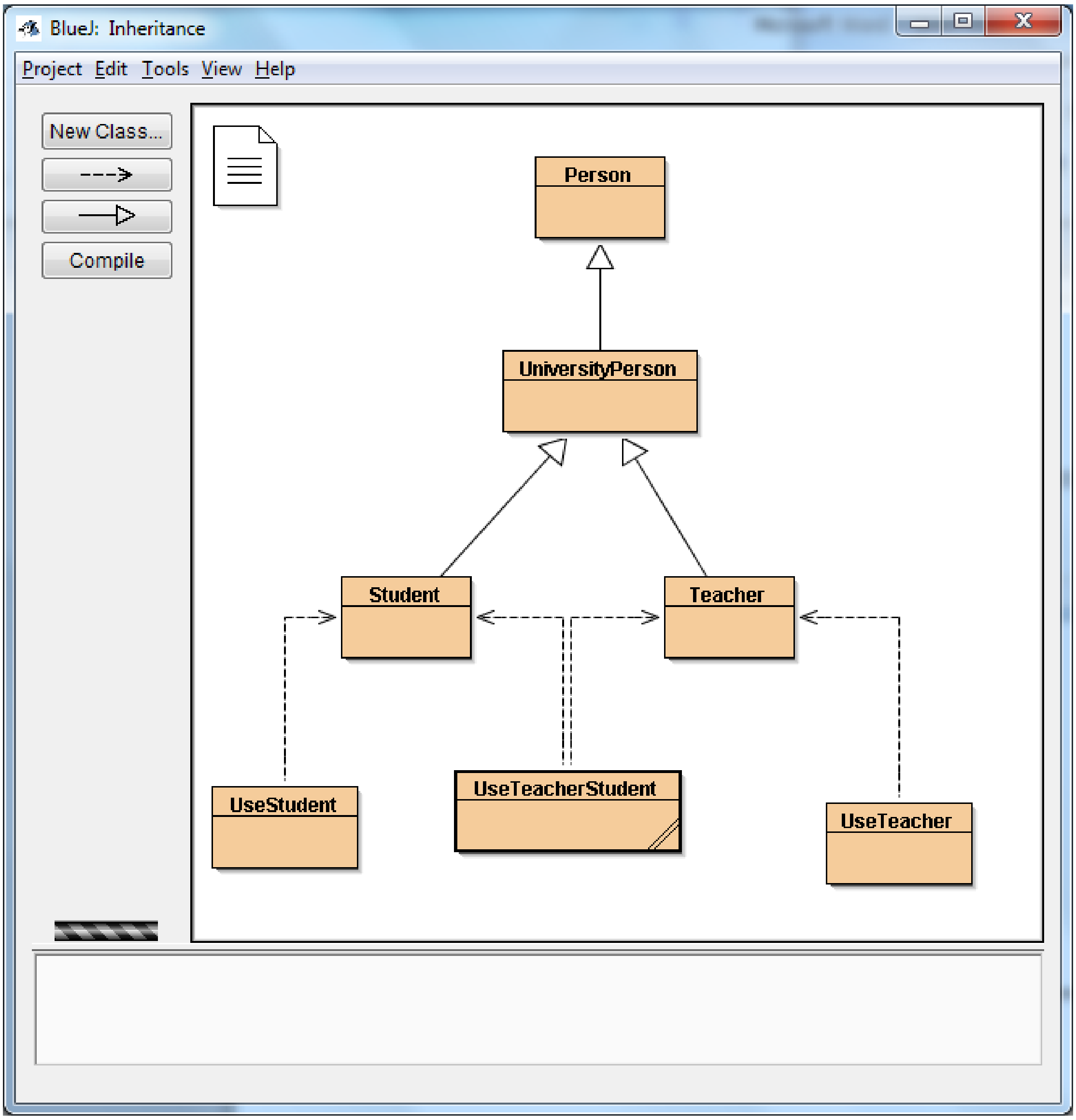
       tom.printDetails();
    }
}
```

{

حيث أن النتيجة ستكون :



وأن شكل الفصائل في الـbluej ستكون كالتالي :



تدريبات (4)

1- اكتب برنامج جافا مكون من فصيل واحد اسمه ASCIICode يحتوي فقط الدالة الرئيسية *main()* يطبع القيم الرمزية في جدول الاسكي للأرقام من بين 65 و 128 على ان يتم طباعة كل 5 رموز في سطر.

2- **أكتب برنامج جافا مكون من فصيل واحد اسمه PrimaryNumbers يحتوي فقط الدالة الرئيسية *main()* لإدخال أي عدد صحيح ومن يطبع اذا كان العدد اولي او لا .**

3- **أكتب برنامج جافا مكون من فصيل واحد اسمه Exercise يحتوي فقط الدالة الرئيسية *main()* لحساب المتوالية التالية :**

$$y = \frac{1}{(x)} + \frac{2}{(x-1)^2} + \frac{3}{(x-1)^3} + \frac{4}{(x-1)^4} + \ \dots \ + \frac{n}{(x-1)^n}$$

حيث N و X مدخلة من الشاشة.

اجابة تدريبات (4)

1- الفصيل ASCIICode :

```
public class ASCIICode
{
    public static void main(String[] args)
    {

        System.out.print('\u000C');

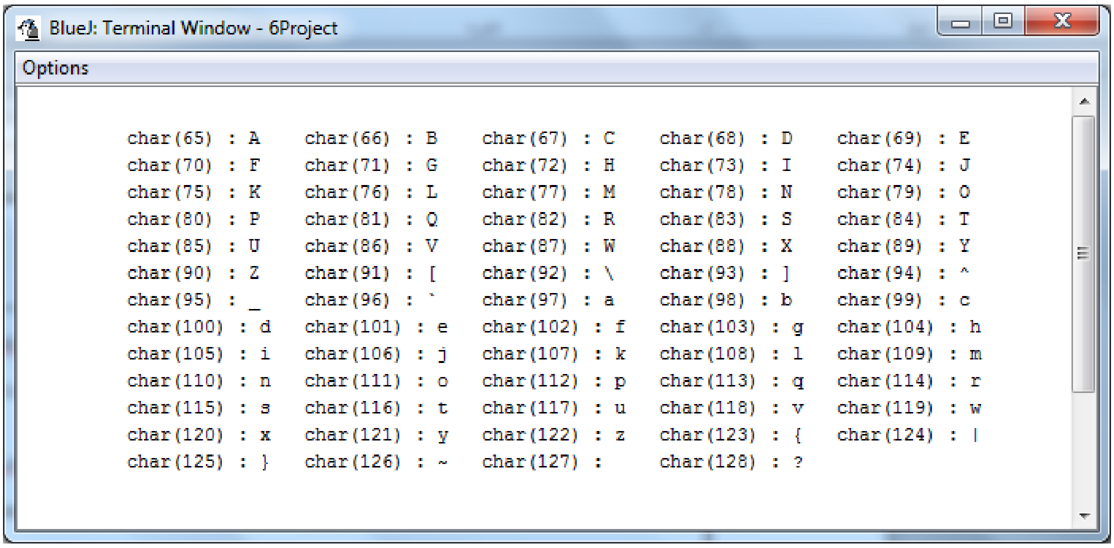
        for (char ch = 65;  ch<=128;   ch++)
        {
            if (ch % 5 ==0)
                System.out.print("\n");

            System.out.print(" \t char(" + (int)ch + ") : "+ ch);

         }

    }

}
```



2- الفصيل PrimaryNumbers :

```
import java.util.Scanner;

public class PrimaryNumbers
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

        System.out.print("Enter Integer Number : ");
        int n = kb.nextInt();

        boolean isPrimary = true;
        int k = 2;

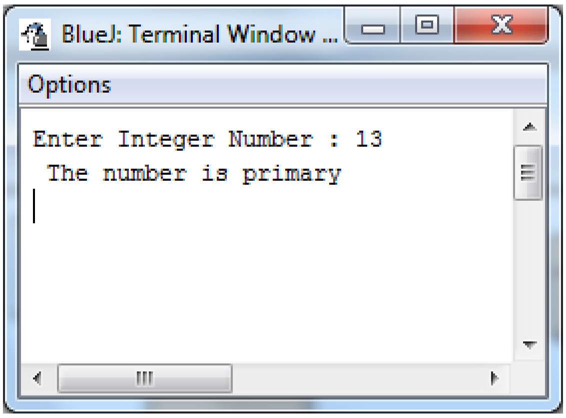
        while ( (k < n) && (isPrimary) )
        {
             if (n % k == 0 )
             {
                 isPrimary = false;
                 break;
              }

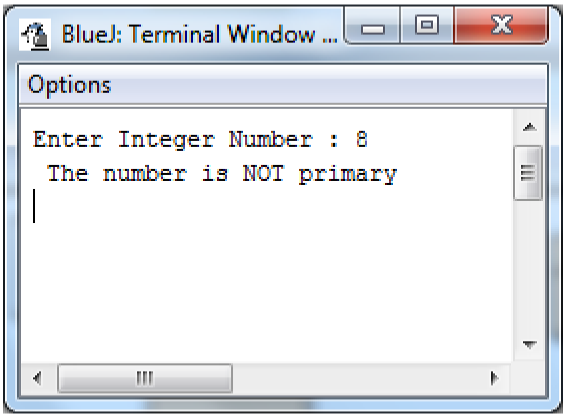
               k++;
         }

         if ( isPrimary )
              System.out.println(" The number is primary");
         else
              System.out.println(" The number is NOT primary");

    }
}
```

العدد الاولي هو العدد الذي يقبل القسمة على نفسه او الواحد الصحيح ، لذلك كان ابتداء العداد k من 2 الى قيمة (العدد -1) أي (k<n) ، فاذا قبل القسمة على أي عدد اصغر منه بدون باقي (n%k==0) فيعني ان العدد غير اولي ويتم الخروج من الحلقة باستخدام جملة ***break*** ثم نختبر قيمة المتغير المنطقي ***isPrimary*** والذي بالافتراض وضعناها ***true*** ، فاذا لم تتغير داخل الحلقة فيعني ان العدد اولي ، اما اذا تغيرت داخل حلقة ***for*** الى ***false*** فيعني ان العدد غير اولي .





3- الفصيل Exercise :

$$y = \frac{1}{(x)} + \frac{2}{(x-1)^2} + \frac{3}{(x-1)^3} + \frac{4}{(x-1)^4} + \dots + \frac{n}{(x-1)^n}$$

```
import java.util.Scanner;

public class Exercise
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print('\u000C');

        System.out.print("Enter Number X, N : ");
        double x = kb.nextInt();
        int    n = kb.nextInt();


        double y = 1/x;

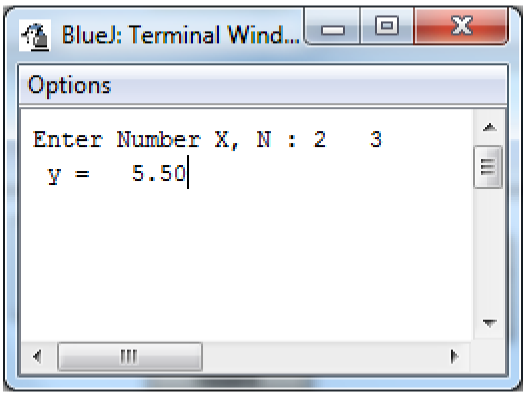
        for (int i = 2; i<=n ; i++)
        {

               y = y + ( i / Math.pow(x-1, i) );
         }

         System.out.format(" y = %6.2f" , y);
    }

}
```

لاحظ ان البسط هو 1,2,3,…,n فيمثله i ، ايضا لاحظ ان الاساس في المقام هو (x-1) عدا اول حد وان الاس او القوة هي 1,2,3,…,n والذي يمثله i ايضا . لاحظ ايضا ان القيمة المبدئية لـ y هي 1/x لان هذا الحد يختلف عن بقية الحدود.



تمرين (2)

1- اكتب برنامج جافا مكون من فصيل اسمه ArithmeticOp يحتوي فقط الدالة الرئيسية main() والتي تقوم بقراءة قيمتين من الشاشة (a, b) ثم ايجاد وطباعة قيمة (x, y) حسب الاتي :

اذا كانت a<b فان :

$$y = a + b$$
$$x = a - b$$

اما اذا كانت a>=b فان :

$$y = a*b$$
$$x = a/b$$

2- اكتب برنامج جافا مكون من فصيل اسمه Equation يحتوي فقط الدالة الرئيسية main() ومستخدما switch لحساب قيمة y كالتالي :

$$y = \begin{cases} x^6 - 4x + 8 & \text{if } x = 2, 8 \\ \sqrt[2]{x-2} + 15 & \text{if } x = 27, 18, 11 \\ x^2/(2x-3)+|x| & \text{if } x = 6, 7 \\ 0 & \text{if } x = \text{عدا ما ذكر اعلاه} \end{cases}$$

3- أكتب برنامج جافا مكون من فصيل اسمه Stars يحتوي فقط الدالة الرئيسية main() لطباعة الشكل التالي لعدد من السطور N حيث ان N هو عدد مدخل من الشاشة يمثل عدد السطور :

```
Enter Integer Number : 7

          *
         ***
        *****
       *******
      *********
     ***********
    *************
```

4- أكتب برنامج جافا مكون من فصيل اسمه **Equation** يحتوي فقط الدالة الرئيسية **main()** لحساب المتوالية التالية :

$$y = \frac{2}{(x+1)^3} + \frac{4}{(x+2)^6} + \frac{6}{(x+3)^9} + \frac{8}{(x+4)^{12}} + \ \dots \ + \frac{2n}{(x+n)^{3n}}$$

حيث **N** و **X** مدخلة من الشاشة.

5- اكتب برنامج جافا مكون من فصيل اسمه **Exercise** يحتوي فقط الدالة الرئيسية **main()** لطباعة الأعداد من بين عددين مدخلين من لوحة المفاتيح وطباعة مربعاتها و مجموع الأعداد ومربعاتها باستخدام حلقة **for**.

6- اكتب برنامج جافا مكون من فصيل اسمه **Equation** يحتوي فقط الدالة الرئيسية **main()** لحل المعادلة :

$$P = \frac{n!}{(n-m)!}$$

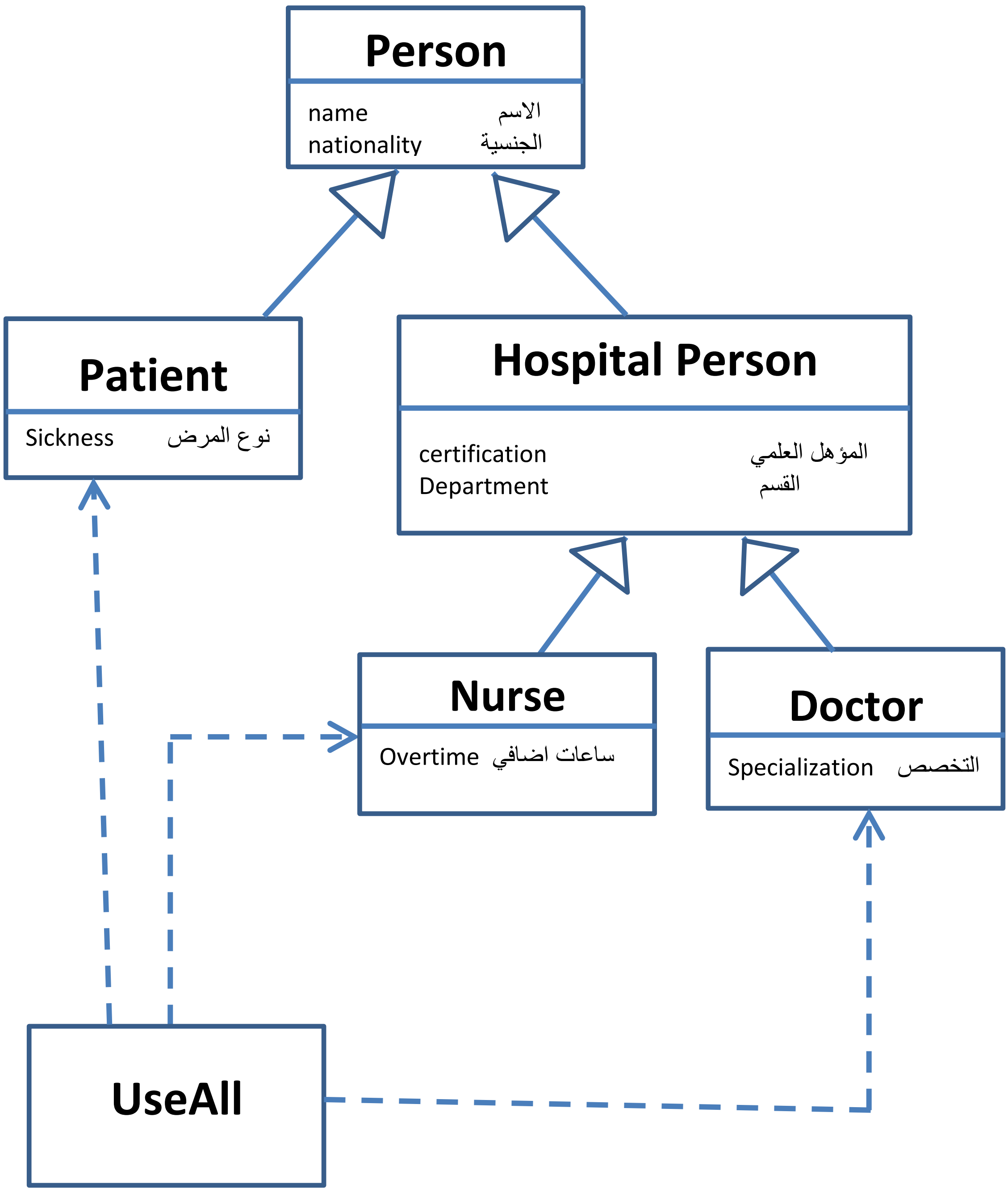
7- اكتب برنامج جافا مكون من فصيل اسمه **MinAndMax** يحتوي فقط الدالة الرئيسية **main()** لإيجاد اكبر وأصغر عدد من بين 10 قيم صحيحة مدخلة من لوحة المفاتيح.

8- مستخدما مفاهيم البرمجة الشيئية قد المستطاع ، و مستخدما مفهوم الوراثة كون فصائل يمثل مخطط الفصائل التالي مع تكوين دالة طباعة في كل فصيل اساسي والفصائل الفرعية، ايضا كون فصيل اسمه **UseAll** لتكوين كائنات من كل فصيل كونته يقوم بطباعة حقوله.

والفصائل هي :

- فصيل الشخص **Person** .
- فصيل المريض **Patient** .
- فصيل شخص المستشفى **HospitalPerson** .
- فصيل الدكتور **Doctor** .
- فصيل الممرضة **Nurse** .

والحقول الخاصة بكل فصيل مبينة بالمخطط التالي :


Person
name الاسم
nationality الجنسية
Patient
Sickness نوع المرض
Hospital Person
certification المؤهل العلمي
Department القسم
Nurse
Overtime ساعات اضافي
Doctor
Specialization التخصص
UseAll

المصفوفات Arrays

استخدمنا في برامج الجافا السابقة أسماء متعددة لتعريف الحقول والمتغيرات ، ولكن ماذا لو اردنا ان نستخدم في احدى البرامج 500 حقل مثلا او اكثر، لغة الجافا مثل اغلب اللغات توفر لنا ما يسمى بالمصفوفات لكي نستخدم اسم واحد لمجموعة من المتغيرات ويتم التمييز بينها باستخدام رقم الفهرس index للوصول الى العنصر المطلوب. سنقوم بدراسة نوعين من المصفوفات وهي : المصفوفات احادية البعد، والمصفوفات ثنائية البعد ، كما يمكن تكوين المصفوفات ذات ابعاد اكثر بنفس الطريقة.

* المصفوفات أحادية البعد One Dimensional Array :

هي مجموعة من العناصر من نفس النوع يتم تخزينها في صف او عمود ذات طول ثابت لها اسم واحد ويتم الوصول الى اي عنصر باستخدام اسم المصفوفة وترتيب العنصر داخل المصفوفة او ما يسمى بفهرس او دليل المصفوفة. العناصر داخل المصفوفة قد تكون من النوع الاساسي Primitive Type او مرجع الى كائن Object .

الشكل العام لتعريف مصفوفة احادية البعد هي :

```
Type[] arrayName;
```

حيث أن :

Type : نوع بيانات العناصر التي سيتم تخزينها بالمصفوفة مثلا : (**int, double, boolean, …etc**).

arrayName : هو اسم المصفوفة وهو مرجع لكائن حتى وان كانت العناصر المخزنة به من النوع الاساسي primitive.

او يمكن تعريفها كالتالي :

```
Type arrayName[];
```

ولكن لا شيء موجود حتى الان حتى بناء المصفوفة باستخدام ***new*** كالتالي :

```
Type[] arrayName = new Type[integer-expression];
```

فمثلا لتعريف مصفوفة اسمها a تتكون من خمس اعداد صحيحة يمكننا كتابة التالي :

```
int[] a = new int[5];
```

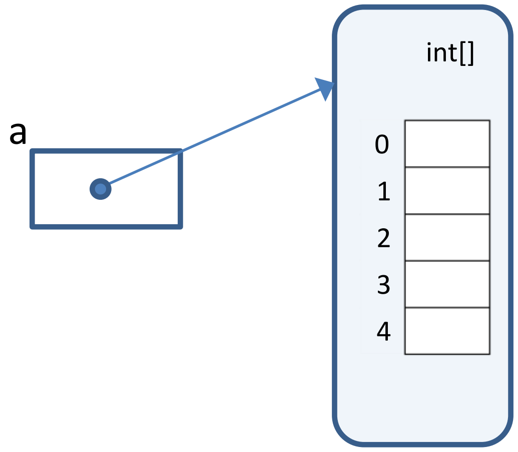
او كالتالي :

```
int[] a;
a = new int[5];
```

هنا سيتم تكوين مصفوفة احادية البعد تحتوي على خمس عناصر صحيحة وسيتم تمهيد العناصر واعطاءها قيمة صفر تلقائيا.

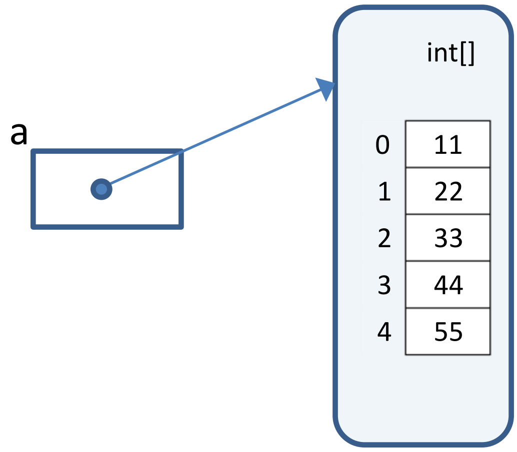
اما اذا كانت العناصر حقيقية فسيتم اعطاءها 0.0 ، وفي حالة ما اذا كانت العناصر منطقية فسيتم تعبئتها بـfalse. اما اذا كانت مصفوفة من الكائنات فسيتم تعبئتها بـnull تلقائيا عند تكوينها.

فتعريف المصفوفة السابق يكون كالتالي :



لاحظ ان فهرس المصفوفة في الجافا مثل لغة السي تبدأ من صفر وينتهي بطول المصفوفة -1 .

فاذا تمت تعبئة المصفوفة بالقيم الصحيحة سيكون شكلها كالتالي :



فللوصول الى العنصر الثالث (العنصر ذو الفهرس 2) واسنادها الى المتغير الصحيح k نكتب :

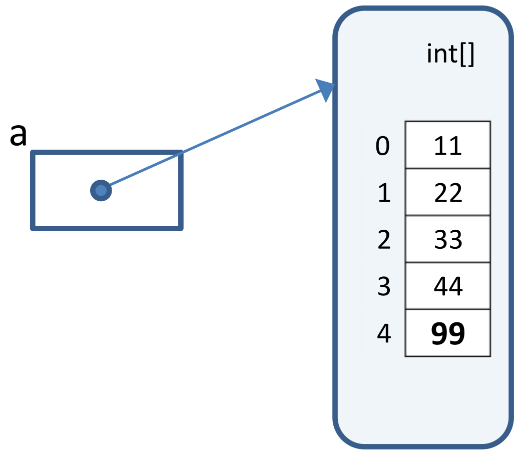
```
int k = a[2];
```

المتغير k الان سيحتوي على القيمة 33.

اما لو اردنا تخصيص قيمة لعنصر من عناصر المصفوفة فنكتب :

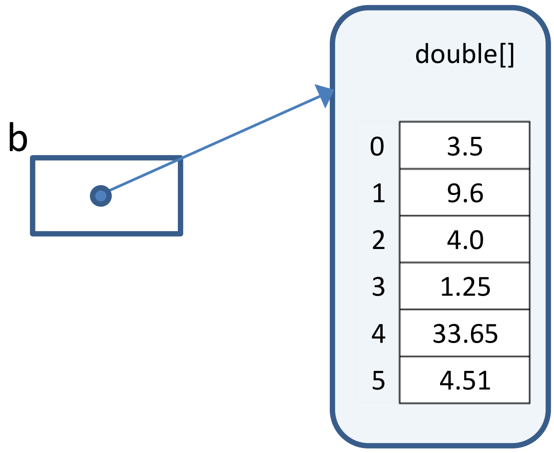
```
a[4] := 99;
```

يعني خزن القيمة 99 في موقع المصفوفة ذو الفهرس 4 ، فتصبح كالتالي :



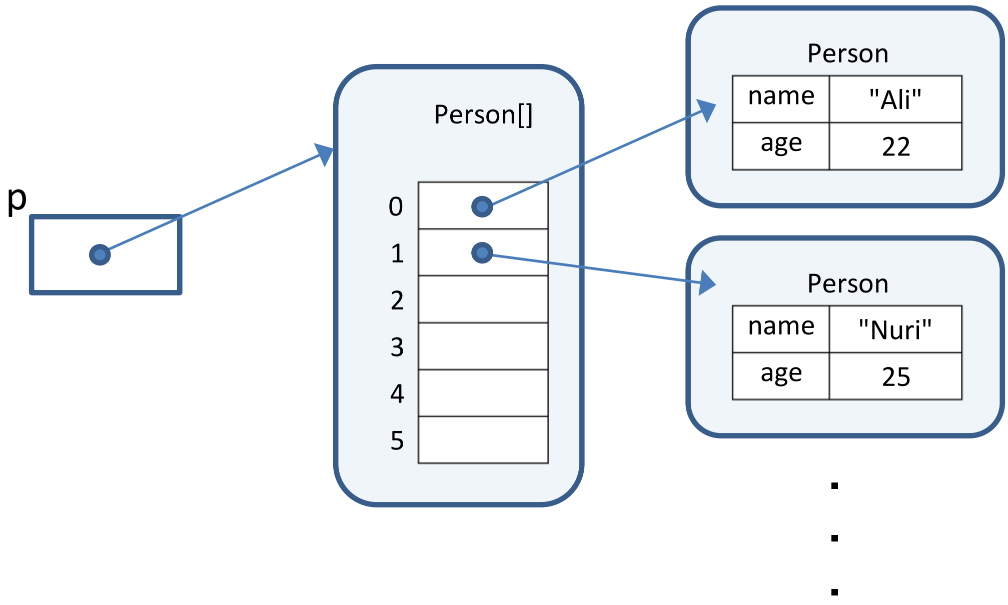
ولتعريف مصفوفة من النوع الحقيقي لها 6 عناصر يكون كالتالي :

```
double[] b = new double[6];
```



اما لتعريف مصفوفة من الكائنات (لنفترض ان لدينا فصيل الشخص Person السابق دراسته بمثال الوراثة والذي يحتوي على حقلي الاسم name و العمر age ونريد تكوين مصفوفة من كائنات الشخص) فيتم كالتالي :

```
Person[] p = new Person[6];
```



فللوصول الى اسم الشخص الثاني (العنصر ذو الفهرس 1) واسنادها الى متغير من نوع السلسلة الرمزية name نكتب :

```
String name = p[1].getName();
```

هنا المتغير name سيصبح لها القيمة "*Nuri*".

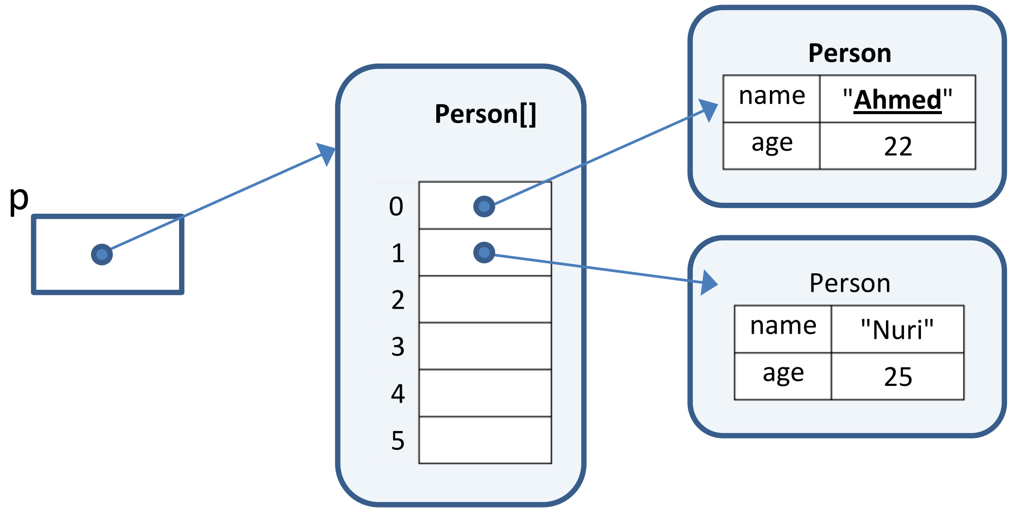
أما للوصول الى عمر الشخص الاول (ذو الفهرس 0) واسنادها الى المتغير ag :

```
int ag = p[0].getAge();
```

اما لتخزين الاسم "*Ahmed*" في الشخص الاول بالمصفوفة (ذو الفهرس 0) فنكتب :

```
p[0].setName("Ahmed");
```

لاحظ اننا استخدامنا مناهج الـ get و الـset للوصول/تغيير قيم الحقول ، لان الحقول خاصة وهذه المناهج عامة وتستخدم لتغيير او الوصول الى الحقول وبالتالي ستكون النتيجة هي :

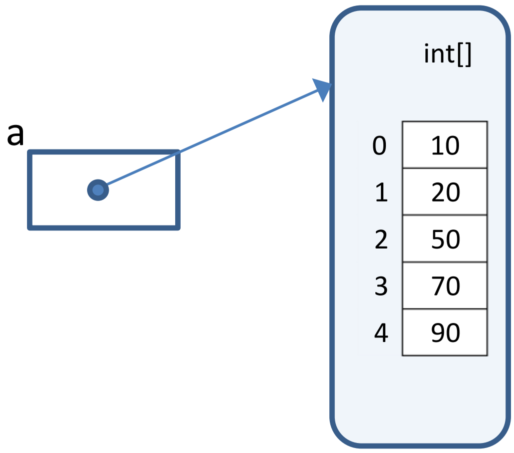


ملاحظة : كل المصفوفات لها حقل صحيح عام اسمه length يقوم بترجيع طول المصفوفة.

كما يمكن تعريف المصفوفات بطريقة اخرى و هي اعطاءها قيم مبدئية عند تعريفها كالتالي :

```
int[] a = {10, 20, 50, 70, 90};
```

هنا قمنا بتعريف وتخصيص قيم مبدئية للمصفوفة في نفس الوقت ، المترجم سوف يحسب عدد عناصر المصفوفة تلقائيا ومن ثم يحدد حجم المصفوفة ، وفي حالتنا هذه 5 عناصر كالتالي :



لاحظ اننا نقوم بفصل قيم عناصر المصفوفة عن بعضها بالفاصلة ، وتكون القيم محصورة بين اقواس الفئة { } .

كما يمكن مثلا تعريف مصفوفة من السلاسل الرمزية وتمهيدها عند التعريف كالتالي :

```
String[] monthNames =
                    {"Jan" , "Feb" , "Mar" , "Apr"
                     "May" , "Jun" , "Jul" , "Aug"
                     "Sep" , "Oct" , "Nov" , "Dec"};
```

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Array1*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال خمسة عناصر صحيحة في مصفوفة احادية وطباعتها.

```
import java.util.Scanner;

public class Array1
{

    public static void main(String[] args)
    {
        int[] a = new int[5];

        Scanner kb = new Scanner(System.in);

        System.out.println ('\f');

        for(int i = 0 ; i< a.length ; i++)
        {
            System.out.print("Enter A[" + i + "] ==>> " );
            a[i] = kb.nextInt();
        }

        System.out.println("------------------");

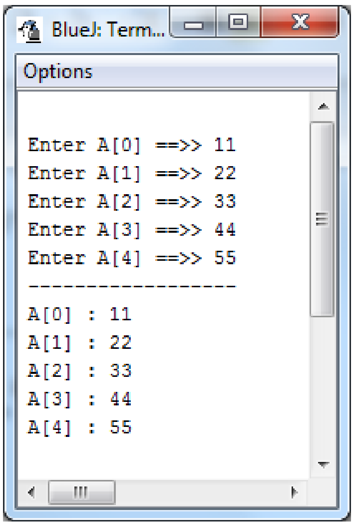
        for(int i = 0 ; i< a.length ; i++)
            System.out.println ("A[" + i + "] : " + a[i]);

    }

}
```

القراءة أو الادخال في مصفوفة

طباعة عناصر مصفوفة



لاحظ اننا استخدمنا a.length والذي يقوم بإرجاع طول المصفوفة ، ايضا استخدمنا حلقة for للمرور على عناصر المصفوفة واحد تلو الاخر .

بدل من استخدام العدد 5 لتحديد طول المصفوفة ، يمكن استخدام ثابت لتحديد طول المصفوفة كالتالي :

```
final int MAX = 5;

int[] a = new int[MAX];
```

كما يمكننا ابتداء من جافا 5 استخدام حلقة for المحسنة (Enhanced for) والتي تأخذ الشكل التالي :

```
for (Type name : arrayName)
{
        عدة جمل ;
}
```

حيث ان :

Type : هو نوع نفس نوع بيانات المصفوفة .

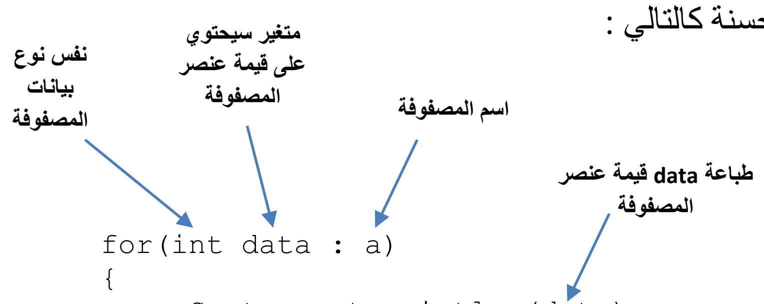
name : اسم متغير وسيحتوي على بيانات عناصر المصفوفة.

arrayName : اسم المصفوفة المراد المرور على عناصرها عنصر تلو الاخر.

فمثلا اذا كانت لدينا حلقة for التالية لطباعة عناصر المصفوفة :

```
for(int i = 0 ; i< a.length ; i++)
{
    System.out.println ( a[i] );
}
```

يمكننا كتابتها بحقلة for المحسنة كالتالي :




```
for(int data : a)
{
     System.out.println (data);
}
```

هنا سيتم المرور على عناصر المصفوفة عنصر عنصرا وفي كل مرة سيحتوي المتغير data على قيمة عنصر المصفوفة فمثلا في الحلقة الاولى سيحتوي على a[0] ، ثم في الحلقة الثانية سيحتوي على a[1] ..... وهكذا.

فمكننا اعادة المثال اسابق كالتالي :

```
import java.util.Scanner;

public class Array11

{

    public static void main(String[] args)

     {
           final int MAX = 5;
           int[] a = new int[MAX];

           Scanner kb = new Scanner(System.in);

           System.out.println ('\f');
```

القراءة أو الادخال في مصفوفة

```
           for(int i = 0 ; i< a.length ; i++)
           {
               System.out.print("Enter A[" + i + "] ==>> " );
               a[i] = kb.nextInt();
           }

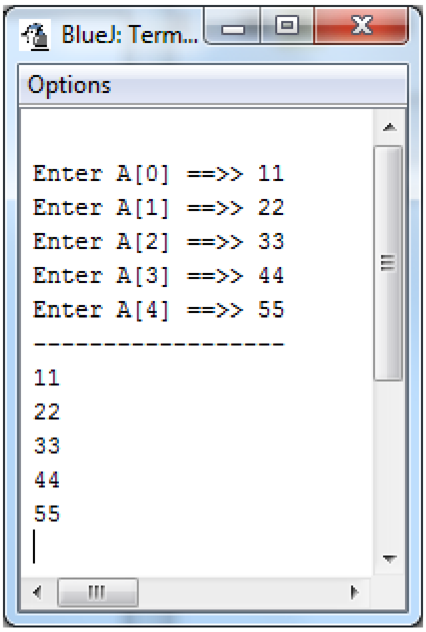
           System.out.println("------------------");
```

طباعة عناصر مصفوفة باستخدام for المحسنة

```
           for(int data : a)
               System.out.println (data);


     }

}
```



اما لطباعة الفهرس مع القيمة في جملة for المحسنة فيمكننا استخدامها كالتالي :

```
int i =0;
for(int data : a)
{
    System.out.println ("A[" + (i++) + "] : " + data);
}
```

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Array*2** يحتوي فقط الدالة الرئيسية ***main***() لإدخال خمسة عناصر من نوع السلسلة الرمزية في مصفوفة احادية ومن ثم ادخال سلسلة اخرى والبحث عنها ضمن عناصر المصفوفة وطباعة فيما اذا كانت موجودة او لا.

```
import java.util.Scanner;
public class Array2
{
    public static void main(String[] args)
    {
        String[] a = new String[5];

        System.out.println ('\f');
        Scanner kb = new Scanner(System.in);

        for(int i = 0 ; i< a.length ; i++)
        {
          System.out.print("Enter String[" + i + "] ==>> " );
          a[i] = kb.nextLine();
        }

        System.out.print("\n Enter a new String : " );
        String lookFor = kb.nextLine();

        boolean isFound = false;
        for(String data : a)
        {
           if (lookFor.equals(data) )
           {
                 isFound = true;
                 break;
           }
        }
      if (isFound)
            System.out.println("The item is found ..." );
      else
           System.out.println("The item is NOT found ..." );
    }
}
```

القراءة أو الادخال في مصفوفة

السلسلة المراد البحث عنها

حلقة للبحث عن السلسلة ضمن المصفوفة وعند وجودها يتم الخروج من الحلقة مباشرة

طباعة هل السلسلة موجودة او لا

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Array3*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال ستة عناصر حقيقة تمثل اسعار سلع في مصفوفة احادية وايجاد مجموعها وطباعة الاسعار عكسيا.

```
import java.util.Scanner;

public class Array3
{
    public static void main(String[] args)
    {
         final int MAX = 6;
         double[] price = new double[MAX];

         Scanner kb = new Scanner(System.in);

         System.out.println ('\f');
```

القراءة أو الادخال في مصفوفة

```
         for(int i = 0 ; i< price.length ; i++)
         {
            System.out.print("Enter Price[" + i + "] ==>> " );
            price[i] = kb.nextDouble();
         }
```

المعالجة / ايجاد مجموع الاسعار

```
         double sum = 0.0;
         for(int i = 0 ; i< price.length ; i++)
               sum += price[i];

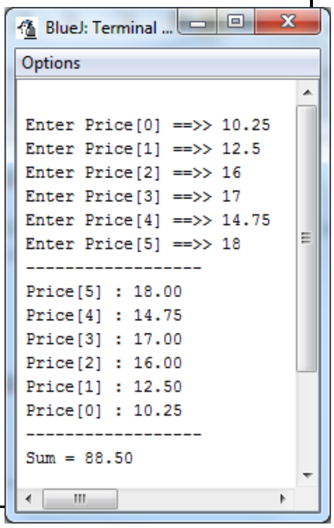
         System.out.println("------------------");
```

الطباعة عكسيا

```
         for(int i = price.length-1 ; i>=0 ; i--)
         {
           System.out.format("Price[%d]:%.2f\n",i, price[i]);
         }
```

طباعة المجموع

```
          System.out.println("------------------");
          System.out.printf("Sum = %5.2f" , sum);
    }
}
```



لاحظ ان برامج المصفوفات عادة تتكون من ثلاثة اجزاء رئيسية هي الادخال و المعالجة والطباعة .

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Array4*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال ستة عناصر صحيحة في مصفوفة احادية وطباعة العناصر الزوجية فقط

```
import java.util.Scanner;

public class Array4

{

    public static void main(String[] args)

    {
          final int MAX = 6;
          int[] a = new int[MAX];

          Scanner kb = new Scanner(System.in);

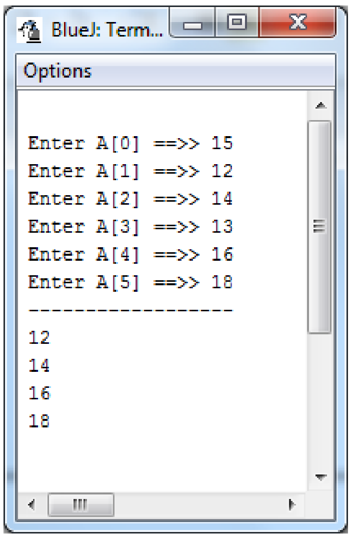
          System.out.println ('\f');

          for(int i = 0 ; i< a.length ; i++)
          {
              System.out.print("Enter A[" + i + "] ==>> " );
              a[i] = kb.nextInt();
          }

          System.out.println("------------------");


          for(int data : a)
          {
              if (data % 2 == 0)
                  System.out.println (data);
           }
    }
}
```

اذا قبل العنصر القسمة على 2 بدون باقي فيعني انه عدد زوجي



مثال :

اكتب برنامج جافا مكون من فصيلين الاول اسمه الشخص ***Person*** يحتوي على حقلي رقم *no* واسم *name* الشخص، كما يحتوي على باني واحد يستقبل قيمتين مناظرتين للحقليين ، ايضا يحتوي على مناهج الـ get و الـ set لكلا الحقلين .اما الفصيل الثاني فاسمه ***ArrayOfPersons*** يقوم بتكوين مصفوفة من كائنات الاشخاص حجمها 3 اشخاص ، ويحتوي على الباني يقوم بقراء قيم تمثل ارقام واسماء الاشخاص الثلاثة وتخزينها بالمصفوفة السابقة الذكر ، ومن ثم طباعة البيانات المدخلة.

الفصيل الاول Person :

```
public class Person
{

     private int no;
     private String name;


     public Person(int n1, String n2)
     {
         no = n1;
         name = n2;

     }

     public String getName()
     {
         return name;
     }

     public int getNo()
     {
         return no;
     }

     public void setName(String n)
     {
          name = n;
     }

     public void setNo(int n)
     {
        no = n;
     }

}
```

الفصل الثاني ArrayOfPersons :

```
import java.util.Scanner;

public class ArrayOfPersons
{

    final int MAX = 3;

    Person[] people = new Person[MAX];

    Scanner kb = new Scanner(System.in);

    public ArrayOfPersons()
    {
        System.out.println ('\f');

        for (int i =0; i < MAX ; i++)
        {
            people[i] =  null;
        }

        for (int i =0; i < MAX ; i++)
        {
            System.out.println("Enter Person["+ (i+1) +"]:" );

            System.out.print("\t\tEnter Number ==>> " );
            int no = kb.nextInt();
            kb.nextLine();

            System.out.print("\t\tEnter Name ==>> " );
            String name = kb.nextLine();

            Person p = new Person(no, name);

            people[i] = p;
        }

        for (int i =0; i < MAX ; i++)
        {
            System.out.format("Number = %d \t Name = %s \n"
                  ,people[i].getNo() , people[i].getName());
        }

    }

}
```

تعريف مصفوفة الاشخاص خارج الباني

تصفير عناصر المصفوفة

ادخال بيانات الشخص من الشاشة

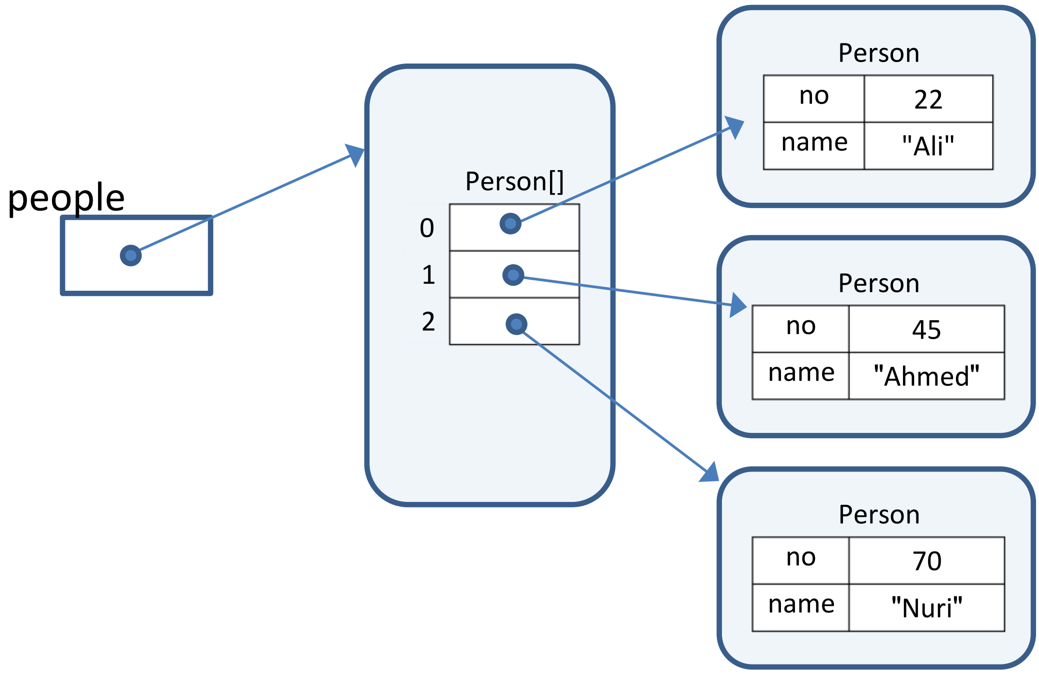
تكوين كائن شخص جديد من البيانات المدخلة

جعل عنصر المصفوفة يشير الى الكائن الجديد

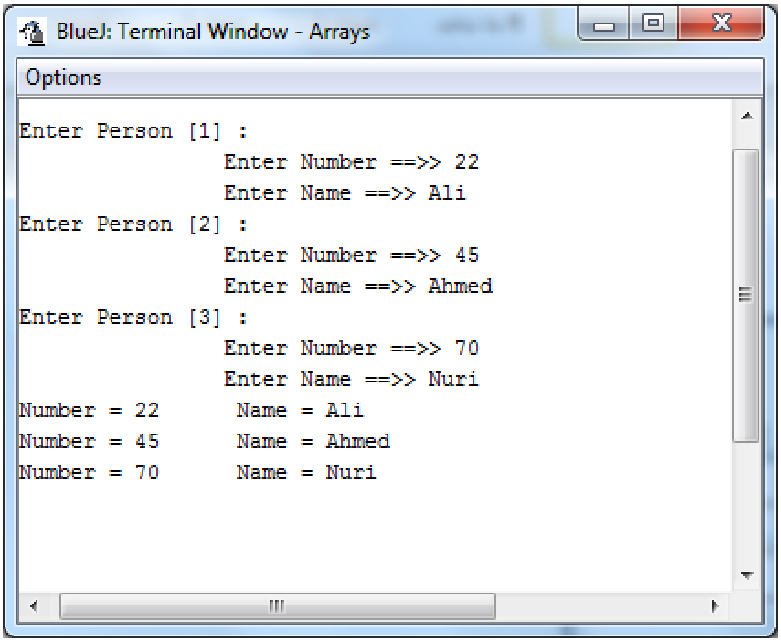
طباعة رقم الشخص والمخزن بالمصفوفة في الموقع i

طباعة اسم الشخص والمخزن بالمصفوفة في الموقع i

المصفوفة السابقة يمكن تمثيلها كالتالي :



وعند التنفيذ ستكون النتيجة كالتالي :



(لاحظ ان البرمجة السابقة ستغنيك عن انشاء مصفوفتين منفصلتين الاولى صحيحة تمثل ارقام الاشخاص والثانية مصفوفة من السلاسل الرمزية تمثل الاسماء).

```
final int MAX = 3;
int numbers = new int[MAX];
String names = new String[MAX];
```

* المصفوفات ثنائية البعد Two Dimensional Array :

في المصفوفات الثنائية البعد يتم تخزين البيانات على هيئة صفوف واعمدة ، ويمكن الوصول الى اي عنصر من عناصر المصفوفة عن طريق رقم الصف ورقم العمود.

الشكل العام لتعريف مصفوفة ثنائية البعد هي :

```
Type[][] matrixName = new Type[int-exp][int-exp];
```

فلتعريف مصفوفة اسمها a مكونة من ثلاثة صفوف وأربعة اعمدة من الاعداد الصحيح نكتب :

```
int[][] a = new int[3][4];
```

بصورة بسيطة يمكن تخيلها كالتالي :

a	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

الاعمـــــــــــــــــــدة

الصفـــــوف

a	0	1	2	3
0	11	22	33	44
1	55	66	77	88
2	99	10	20	30

محتويات المصفوفة (القيم)

فللوصول الى العنصر في الصف رقم (1) والعمود رقم (2) وتخزينه في متغير اخر اسمه k نكتب :

```
k := a[1][2];
```

يعني ان k الآن يحتوي على القيمة 77.

اما لو اردنا ان نخصص قيمة لعنصر من عناصر المصفوفة فنكتب :

```
a[2][3] := 40;
```

يعني خزن القيمة 40 في موقع المصفوفة ذو الصف رقم 2 والعمود رقم 3 ، فتصبح كالتالي :

	0	1	2	3
0	11	22	33	44
1	55	66	77	88
2	99	100	120	40

المصفوفة الثنائية البعد يمكن النظر اليها على انها مصفوفة احادية ، وكل خلية بها هي مصفوفة احادية اخرى.

فاذا كان لدينا التعريف التالي :

```
int[][] a = new int[3][4];
```

فانه يمكننا ايجاد عدد الصفوف بالمصفوفة كالتالي :

```
a.length
```

والتي سترجع طول البعد الاول للمصفوفة ، أي 3 ، أما اذا اردنا ايجاد البعد الثاني للمصفوفة (أي عدد الاعمدة) فيمكننا كتابة التالي :

```
a[1].length
```

أي اوجد حجم الصف الاول (بمعنى اخر : عدد الاعمدة بالصف الاول وهي هنا 4).

في المصفوفات الثنائية عادة ما تكون كل الصفوف متساوية الطول كما سبق، ولكن في الجافا يمكن الاعلان عن مصفوفة ذات صفوف مختلفة الطول ، فمثلا يمكن تكوين مصفوفة بشكل مثلثي كالتالي :

```
a[0][0]
a[1][0] a[1][1]
a[2][0] a[2][1] a[2][2]
a[3][0] a[3][1] a[3][2] a[3][3]
a[4][0] a[4][1] a[4][2] a[4][3] a[4][4]
```

فلتعريف مصفوفة كالسابق يمكن كتابة التعريف التالي :

```
int[][] a = new int[5][];
```

اولا : فقط قمنا بتحديد عدد الصفوف كما سبق ولم نحدد الاعمدة (طول كل صف) .

ثانيا : سنقوم بتحديد طول كل صف لوحده كالتالي :

```
for(int i=0 ; i < a.length ; i++  )
   a[i] = new int[i+1];
```

حيث ان الحلقة ستبدأ من 0 الى عدد الصفوف (a.length) ثم في كل صف سنقوم بتحديد طوله (عدد العناصر الذي ستخزن به) باستخدام :

```
a[i] = new int[i+1];
```

فمثلا في الصف الاول (عندما i=0) سيكون طول الصف 1 (i+1 أي 0+1) وهكذا لبقية الصفوف.

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Matrix*** يحتوي فقط الدالة الرئيسية *main()* لإدخال قيم صحيحة في مصفوفة ثنائية البعد 4 X 3 وطباعتها على شكل مصفوفة ثنائية.

```
import java.util.Scanner;
public class Matrix
{
    public static void main(String[] agrs)
    {
        final int ROWS = 3;
        final int COLS = 4;
        int[][] mat = new int[ROWS][COLS];

        Scanner kb = new Scanner(System.in);
        System.out.print ('\u000C');

        for(int i = 0 ; i< ROWS ; i++)
           for (int j =0 ; j < COLS; j ++)
           {
              System.out.print("Enter Mat["+i+ ","  +j+ "]:");
              mat[i][j] = kb.nextInt();
           }

         for(int i = 0 ; i< mat.length ; i++)
         {
             for (int j =0 ; j < mat[i].length; j++)
                  System.out.print ("\t" + mat[i][j] );

             System.out.print('\n');
         }
    }
}
```

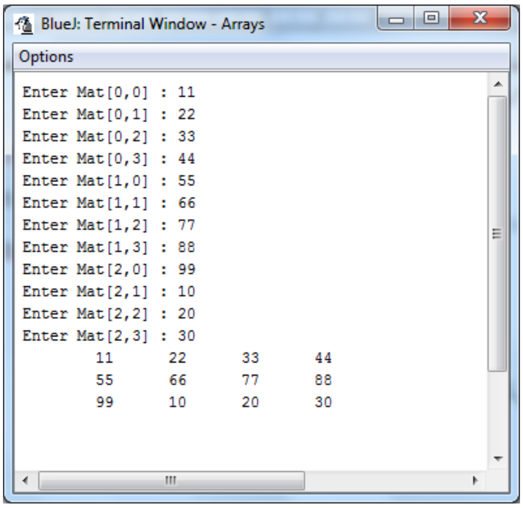
ادخال بيانات مصفوفة

طباعة عناصر المصفوفة

في طباعة عناصر المصفوفة يمكننا استخدام حلقة for السابقة مع حقل الطول *length* او عدد الصفوف والاعمدة مباشرة كالتالي :

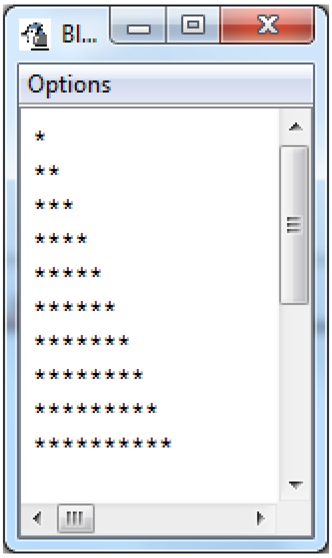
```
for(int i = 0 ; i< ROW ; i++)
{
  for (int j =0 ; j < COLS; j++)
     System.out.print ("\t"+mat[i][j]);
  System.out.print('\n');

}
```



مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***TriangularSahpe*** يحتوي فقط الدالة الرئيسية *main*() لتكوين مصفوفة ثنائية البعد بحجم 10X10 لتخزين رمز الـ'*' على شكل مثلثي بها ومن ثم طباعتها كما بالشكل التالي :




```
import java.util.Scanner;

public class TriangularShape
{
    public static void main(String[] args)
    {
        final int ROWS = 10;
        final int COLS = 10;

        char[][] a = new char[ROWS][];

        Scanner kb = new Scanner(System.in);

        for(int i = 0 ; i< a.length ; i++)
            a[i] = new char [i+1];

         System.out.print('\u000C');

         for(int i = 0 ; i< ROWS ; i++)
             for (int j =0 ; j < a[i].length; j ++)
                 a[i][j] = '*';

          for(int i = 0 ; i< ROWS ; i++)
          {
                for (int j =0 ; j < a[i].length; j ++)
                    System.out.print (a[i][j]);
                System.out.print ("\n");
           }
    }
}
```

عند تكوين المصفوفة قمنا فقط بتحديد عدد الصفوف وليست الاعمدة

هنا قمنا بتحديد طول كل صف (عدد الاعمدة بكل صف)

كن حذرا : عند التعامل مع طول كل صف لوحده ، أي عند التعامل مع الاعمدة بكل صف لوحده

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Matrix2*** يحتوي فقط الدالة الرئيسية *main()* لإدخال قيم صحيحة في مصفوفة ثنائية البعد 3 X 3 وتخزين مجموع كل صف في مصفوفة احادية البعد ثم طباعة عناصر المصفوفة الثنائية وبجوارها عناصر المصفوفة الاحادية.

```java
import java.util.Scanner;
public class Matrix2
{
    public static void main(String[] agrs)
    {
        final int ROWS = 3;
        final int COLS = 3;
        int[][] mat = new int[ROWS][COLS];
        int[] arr = new int[ROWS];
        int sum;

        Scanner kb = new Scanner(System.in);

        System.out.print ('\u000C');
```

ادخال بيانات مصفوفة

```java
        for(int i = 0 ; i< ROWS ; i++)
          for (int j =0 ; j < COLS; j ++)
          {
             System.out.print("Enter Mat[" +i+ ","  +j+ "]:");
             mat[i][j] = kb.nextInt();
          }
```

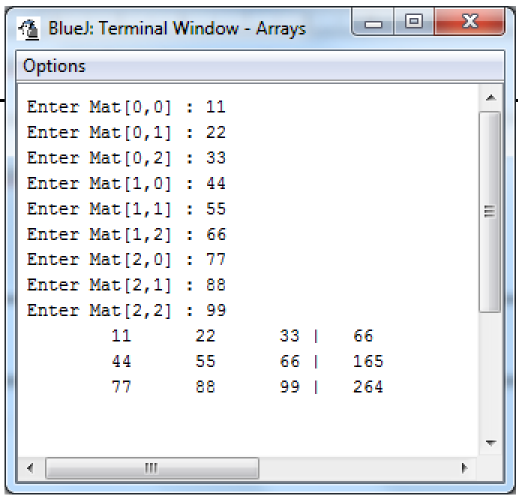
معالجة / ايجاد مجموع كل صف وتخزين في المصفوفة الاحادية

```java
        for(int i = 0 ; i< ROWS ; i++)
        {
            sum = 0;
            for (int j =0 ; j < COLS; j ++)
                 sum = sum + mat[i][j];
            arr[i] = sum;
        }
```

طباعة عناصر المصفوفة الثنائية وبجوارها الاحادية

```java
        for(int i = 0 ; i< mat.length ; i++)
        {
            for (int j =0 ; j < mat[i].length; j++)
                 System.out.print ("\t" + mat[i][j] );
            System.out.print("  |    " + arr[i] +  "\n");
        }
    }
}
```

التنفيذ سيكون كالتالي :



مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Matrix3*** يحتوي فقط على الدالة الرئيسية ***main()*** لإدخال قيم صحيحة في مصفوفتين ثنائية مربعة 3X3 وايجاد حاصل ضرب المصفوفتين.

```
import java.util.Scanner;
public class Matrix3
{
    public static void main(String[] agrs)
    {
        final int ROWS = 3;          final int COLS = 3;
        int[][] a = new int[ROWS][COLS];
        int[][] b = new int[ROWS][COLS];
        int[][] c = new int[ROWS][COLS];

        Scanner kb = new Scanner(System.in);

        System.out.print ('\u000C');
```

ادخال المصفوفة الأولى a

```
        for(int i = 0 ; i< ROWS ; i++)
           for (int j =0 ; j < COLS; j ++)
                 a[i][j] = kb.nextInt();
```

ادخال المصفوفة الثانية b

```
        for(int i = 0 ; i< ROWS ; i++)
           for (int j =0 ; j < COLS; j ++)
                 b[i][j] = kb.nextInt();
```

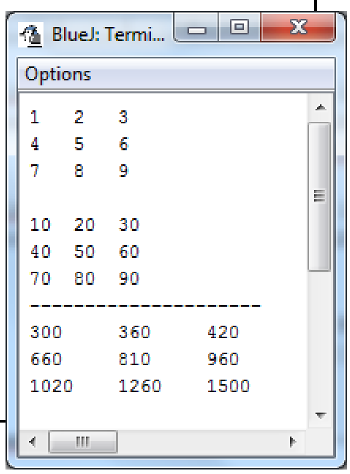
ايجاد حاصل ضرب المصفوفة الاولى في الثانية

```
        for(int i = 0 ; i< ROWS ; i++)
           for (int j =0 ; j < COLS; j ++)
           {
               c[i][j] = 0;
               for (int k =0; k < ROWS; k ++)
                    c[i][j] = c[i][j] + a[i][k] * b[k][j];
           }

        System.out.println ("---------------------");
```

طباعة المصفوفة الجديد c

```
        for(int i = 0 ; i< ROWS ; i++)
        {
            for (int j =0 ; j < COLS; j++)
                 System.out.print (c[i][j] +"\t" );
            System.out.print("\n");
        }
    }
}
```



لاحظ انه عند محاولة الوصول الى عنصر من عناصر المصفوفة خارج حدودها (اكبر من او يساوي طولها) فسوف يحدث استثناء (أي حدوث خطأ عند التنفيذ ***runtime***) فمثلا اذا كان لدينا:

```
int[] a = new int[5];
```

فاذا كتبنا حلقة for للوصول الى عناصر المصفوفة كالتالي :

```
for(int i = 0; i <= 5; i++)
   System.out.println(a[i]);
```

هنا نلاحظ اننا كتبنا (i<=5) في حين ان اخر فهرس في المصفوفة هي 4 فان استثناء سوف يحدث اسمه `ArrayIndexOutOfBoundsException` (أي محاولة الوصول الى فهرس هو خارج حدود المصفوفة).

او عند محاولة القسمة على الصفر مثلا فان استثناء اسمه `ArithmeticException` سوف يحدث فمثلا:

```
Int b = 99/0;
```

يمكننا معالجة الاستثناء باستخدام جملة try….catch كالتالي :

```
try
{
       جمل من المحتمل ان تطلق استثناءات;
}
catch(Exception-name  e)
{
        الاجراء المتخذ عند حدوث الاستثناء ;
}
```

فمثلا اذا كان لدينا التعريف الاتي :

```
int[] a = {10, 20, 30, 40, 50};

try
{
     for(int i = 0; i <= 5; i++)
        System.out.println(a[i]);
}
catch( ArrayIndexOutOfBoundsException  e)
{
       System.out.println("Error: index out of bounds ");
}
```

عند محاولة الوصول الى فهرس عنصر خارج حدود المصفوفة فان الاستثناء سوف ينطلق.

اذا انطلق الاستثناء فان جملة catch سوف تمسكه وتظهر الرسالة هذه الرسالة .

يمكننا كتابة اكثر من جملة catch لالتقاط الاستثناءات كالتالي :

```
int[] a = {10, 20, 30, 40, 50};

try
{
      int b = 99/0;          ← محاولة القسمة على الصفر

      for(int i = 0; i <= 5; i++)     ← محاولة الوصول الى فهرس خارج حدود المصفوفة
          System.out.println(a[i]);
}
catch( ArrayIndexOutOfBoundsException  e)     ← معالجة استثناء محاولة الوصول الى فهرس مصفوفة خارج حدودها
{
        System.out.println("Error: index out of bounds ");
}

catch(ArithmeticException e)     ← معالجة استثناء محاولة القسمة على الصفر
{
        System.out.println("Error : Division by Zero ");
}
catch(Exception e)     ← معالجة أي استثناء اخر لم يتم التقاطه ضمن جمل catch السابقة
{
        System.out.println("Error: Unknown Error ");
}
```

e : هو كائن من نوع الاستثناء يمكن استخدامه لطباعة رسالة تبين نوع الاستثناء الذي حدث باستدعاء المنهج getMessage() كالتالي :

```
try
{
      int b = 50/0;
}
catch(Exception e)     ← كائن من نوع Exception
{
        System.out.println("Error: " + e.getMessage() );
}
```

هذا المنهج سوف يظهر رسالة تبين فحوى الاستثناء الذي حدث وهو هنا :

/ by zero

كما يمكن استخدام المنهج **e.toString()** ليظهر اسم فصيل الاستثناء مع فحوى رسالة تبين ما هو الخطأ الذي حدث.

كما يمكننا استخدام جملة finally والتي ستؤدي الى تنفيذ جمل برمجية ضمنها **عند حدوث او عدم حدوث** أي استثناء ضمن جمل الـtry . فعند استخدام try….catch وحدوث استثناء ضمن احدى جمل الـ try فان التنفيذ سوف يتوقف عند تلك الجملة التي حدث بها الخطأ ويتجه الى الـcatch مباشرة لمعالجة الاستثناء ، اما عند استخدامنا لجملة finally فانه الجمل البرمجية سوف تنفذ ضمنها بغض النظر عن حدوث ومعالجة الاستثناء او عدم حدوثه. والشكل العام لها :

```
try
{
        ;جمل من المحتمل ان تطلق استثناءات
}
finally
{
         ; جمل سيتم تنفيذها سواء حدث استثناء او لم يحدث
}
```

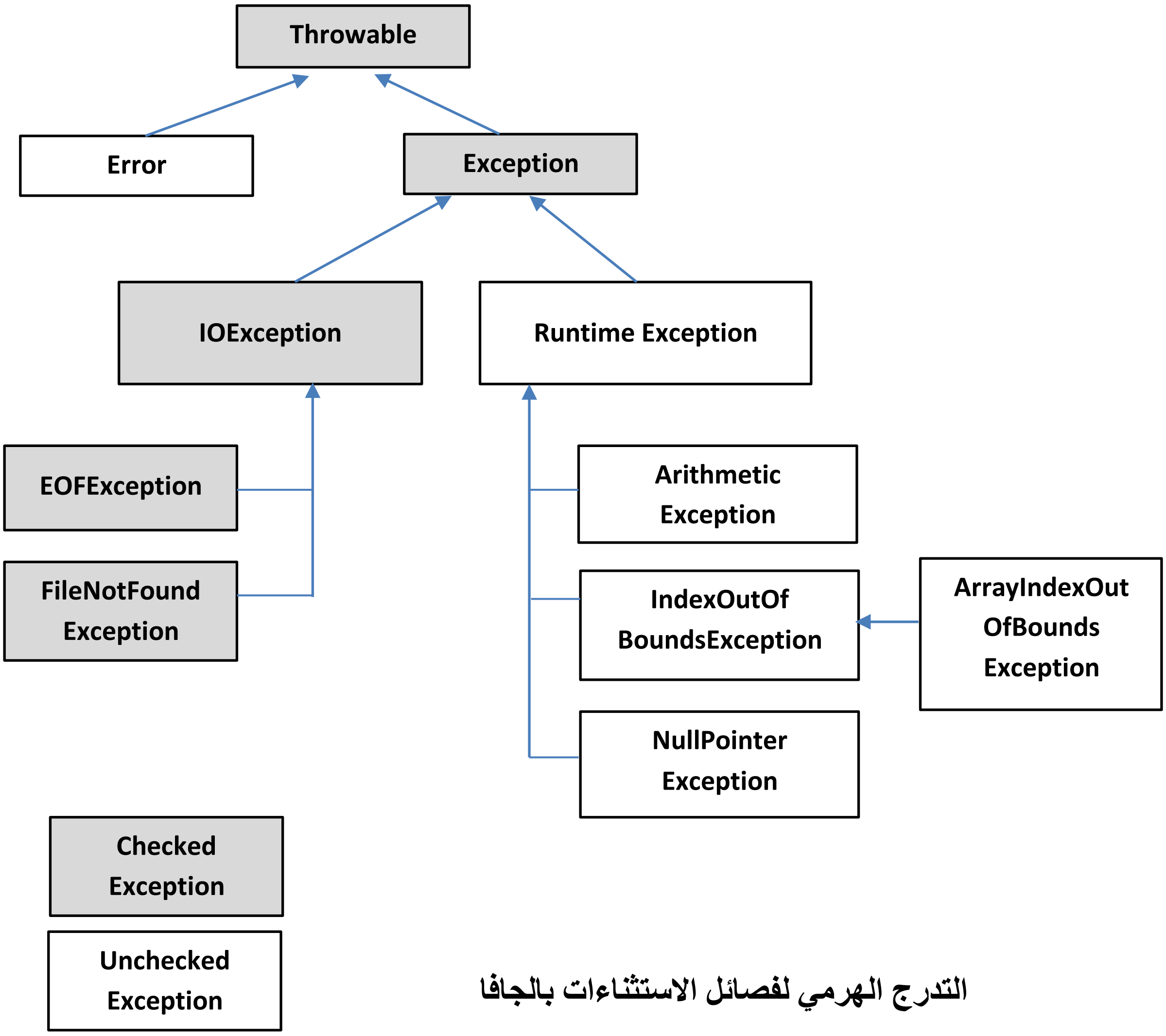
عادة ما يتم استخدام finally للقيام بإجراءات يجب ان تتم قبل انهاء البرنامج سواء حدث الاستثناء (الخطأ الذي سيوقف البرنامج) او لم يحدث . مثل اغلاق الملفات او ترجيع موارد الذاكرة تم حجزها مسبقا قبل انهاء البرنامج سواء حدث الاستثناء او لم يحدث.

يمكن استخدام جملة finally بعد catch في جملة try واحدة .

لا يفوتنا ان نذكر ان الاستثناءات وكما موضح بالشكل التالي يتم تمثيلها ضمن الجافا بفصائل مختلفة ترث الفصيل الاعلى ***Throwable*** وان هناك فصيلتين ترث هذه الفصيلة هما ***Exception*** و ***Error*** وأن الاستثناءات Exception تنقسم الى نوعين هما :

1- Checked Exception : وهي التي لا يمكن اهمالها مثل كل الاستثناءات المتعلقة الادخال والاخراج I/O.
2- Unchecked Exception : وهي الاستثناءات التي ليس من الضروري أن تكون معالجة كاستثناءات وقت التنفيذ (مثلا : `ArithmeticException`).

كما يمكننا تكوين فصائل خاصة بنا للاستثناءات تورث كل خصائص ومناهج فصيل الاستثناء Exception . وبالتالي يمكننا اطلاق هذه الاستثناءات (باستخدام جملة throw) ومعالجتها بجملة catch او تمريرها (باستخدام جملة throws) الى فصائل اخرى لتقوم بمعالجتها . فيمكننا تلخيص ما سبق ان الاستثناء هو خطأ يحدث اثناء تنفيذ البرنامج فيؤدي الى توقف التسلسل الطبيعي للتنفيذ.



التدرج الهرمي لفصائل الاستثناءات بالجافا

مثال :

اكتب برنامج جافا مكون من فصيلين ، الاول اسمه ***MyException*** لتعريف استثنائك الخاص حيث يرث فصيل الاستثناء ***Exception*** ويحتوي على باني واحد يقوم باستقبال رسالة مطلوب اظهارها عند حدوث الخطأ وارسالها الى الفصيل الأعلى ***(Exception)*** عبر الـ***super*** . أما الفصيل الاخر ***UseMyException*** ويحتوي فقط على الدالة الرئيسية ***main()*** يقرأ قيمة صحيحة ويختبرها ، اذا كانت صفر يقوم البرنامج بإطلاق استثنائك الخاص ومعالجة الاستثناء عن طريق التقاطه بـcatch واظهار رسالة الاستثناء ، مع توضيح استخدام جملة finally .

الفصيل الاول ***MyException*** :

```
public class MyException extends Exception
{
    public MyException(String m)
    {
        super(m) ;
    }
}
```

m : رسالة توضح الخطأ وسيتم استقبالها عند حدوث الاستثناء وارسالها الى الفصيل الاعلى عبر الsuper

الفصيل الثاني *UseMyException* :

```
import java.util.Scanner;

public class UseMyException
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);

        try
        {
            System.out.print('\f');

            System.out.print("Enter A :  " );
            int a = kb.nextInt();

            if  (a == 0)
               throw new MyException("No Division by zero ");

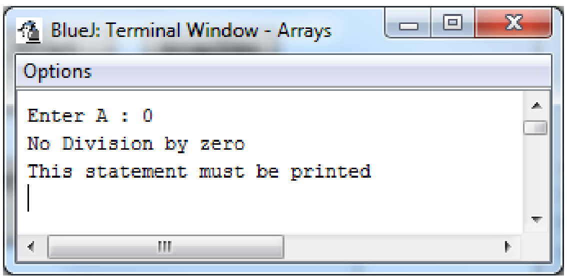
            int b = 20/a;
            System.out.println("Result = " + b);
        }
        catch  (MyException e)
        {
               System.out.println( e.getMessage()  );
        }
        finally
        {
             System.out.print("This statement must be printed");
        }
    }
}
```

هنا يتم اختبار القيمة المدخلة اذا ساوت صفر يتم حذف استثناء باستخدام جملة throw مع ارسال الرسالة المطلوب اظهارها

اذا تم التقاط الاستثناء الذي حذف بواسطة جملة throw السابقة فسوف يتم طباعة الرسالة المرسلة عن طريق المنهج getMessage()

هذه الجملة سيتم طباعتها في جميع الاحوال.

لاحظ انه استخدمنا جملة احذف (**<u>throw</u>**) عندما يتم ادخال قيمة صفر لأنه لا يمكن القسمة على الصفر ، لذلك قمنا بحذف كائن جديد من الاستثناء الذي قمنا بتكوينه باستخدام ***new*** مع ارسال الرسالة التي سوف يتم اظهارها ، هذه الرسالة (*"No Division by zero"*) سيتم استقبالها بواسطة البارامتير ***m*** ضمن باني الفصيل السابق (***MyException***) والذي سيُرسل بدوره الى الفصيل الاعلى (***Exception***) بواسطة ***super*** . ولكون الاستثناء حدث ضمن جملة ***try….catch*** فسنقوم بالتقاط الاستثناء باستخدام جملة ***catch*** ومن ثم طباعة الرسالة باستخدام المنهج ***getMessage()*** . أما الجملة ضمن الـ***finally*** فسيتم طباعتها في جميع الاحوال.

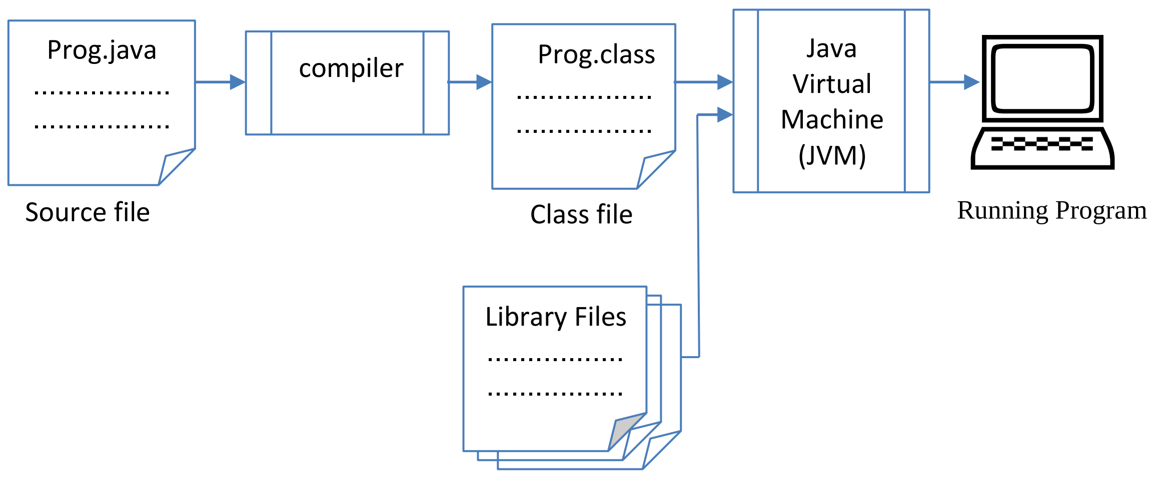


تنفيذ برامج جافا خارج الـBlueJ

الـBlueJ عبارة عن بيئة استخدمناها لكي تساعدنا في تنفيذ برامج جافا بسهولة ، ولكن يمكننا تنفيذ برامج جافا باستخدام أي من البيئات الاخرى مثل (Netbeans او Eclipse او غيرها). او يمكننا تنفيذ برامج الجافا مباشرة باستخدام أي محرر نصوص وتخزينه بامتداد (java.) ومن ثم اجراء عملية الترجمة (compile) باستخدام المترجم ***javac*** لإنتاج ملف بنفس الاسم ولكن بامتداد (class.) والذي يحتوي على تعليمات الالة الوهمية (virtual machine instructions) .

الملف ذو الامتداد (class.) والذي يعرف بالنص البايتي (bytecode) يحتوي فقط على تعليمات برنامجك ، ولكن هذا لا يكفي لتنفيذ البرنامج ، فمثلا لعرض سلسلة رموز على نافذة قد يكون مطلوب بعض الاجراءات الاضافية في المستوي الأدنى (Low-level) لتنفيذ ذلك. حيث قاموا مبرمجون اخرون ببرمجة وترجمة فصيل الـSystem ضمن ملفات المكتبة Library files لكي نقوم بتعريف كائن out ومن ثم استخدام المنهج print لطباعة تلك السلسلة الرمزية.

تقوم آلة جافا الوهمية Java Virtual Machine (JVM) بتحميل تعليمات ملفك (class.) وتحميل ملفات المكتبة (Library files) ومن تم انتاج البرنامج التنفيذي كما موضح بالشكل التالي :



تنفيذ برنامج جافا بداية من كتابة الملف المصدر وصولا الى تنفيذ البرنامج

لكون JVM تكون متاحة على مختلف انظمة التشغيل لذلك فان نفس الملف (class.) يكون قابل للتنفيذ على انظمة التشغيل المختلفة مثل : windows و Linux و Mac ... الخ.

لتكوين برامج جافا يجب اعداد مكتبة او مترجم جافا على جهازك عن طريق تحميل حزمة JDK (Java Development Kit) والتي تمكن تنزيلها من موقع شركة Sun Microsystem مجانا وهو متاح للويندوز بالإصدارين 32bit او 64bit :

http://java.sun.com

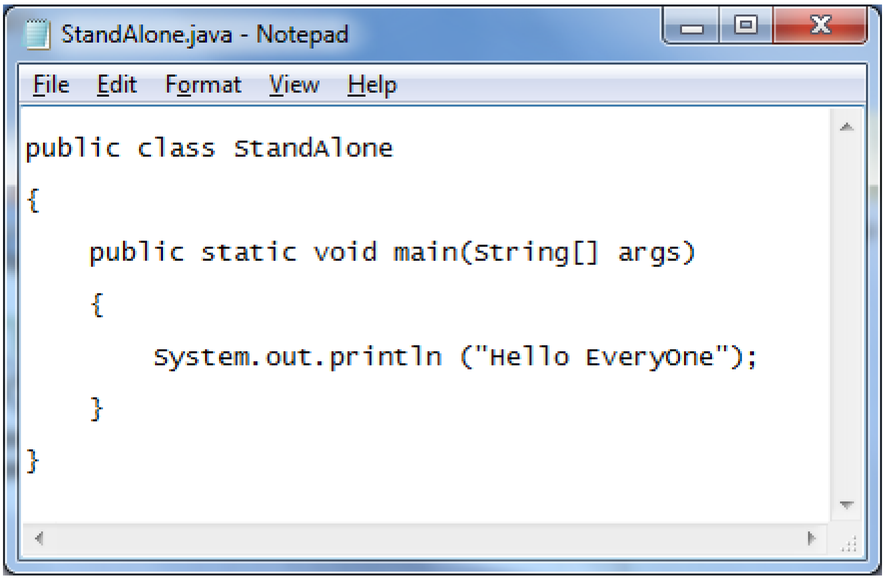
بعد الانتهاء من تحميل الـJDK يكون بإمكاننا تنفيذ برامج جافا فقط من داخل الدليل الفرعي bin ضمن دليل الجافا . ولكن لكي يمكننا تنفيذ برامج جافا من أي دليل يجب اضافته الى ملف مسار النظام (System path) كالتالي :

ضمن ويندوز xp أويندوز 7 او ويندوز vista انتقل الى :

Start ثم الى control Panel ثم الى System ثم الى (system Settings) Advanced ثم الى Environment Variable .

ثم اختار متغير النظام "path" وانقر على زر التحرير Edit واضف الدليل الفرعي bin الخاص بالجافا وليكن مثلا : c:\java\bin; ثم انقر على زر ok .

الان افتح برنامج المفكرة واكتب به الاتي :



قم بتخزينه بنفس اسم الفصيل وبامتداد (.java) أي بالاسم StandAlone.java وخزنه على القرص :F مثلا ، لاحظ انه يجب كتابة الدالة الرئيسية ()main حيث سيبدأ التنفيذ منها.

الان شغل محث الـdos وذلك بكتابة cmd ضمن التشغيل run وانتقل الى القرص :F وذلك بكتابة : (F:) ثم اضغط enter.

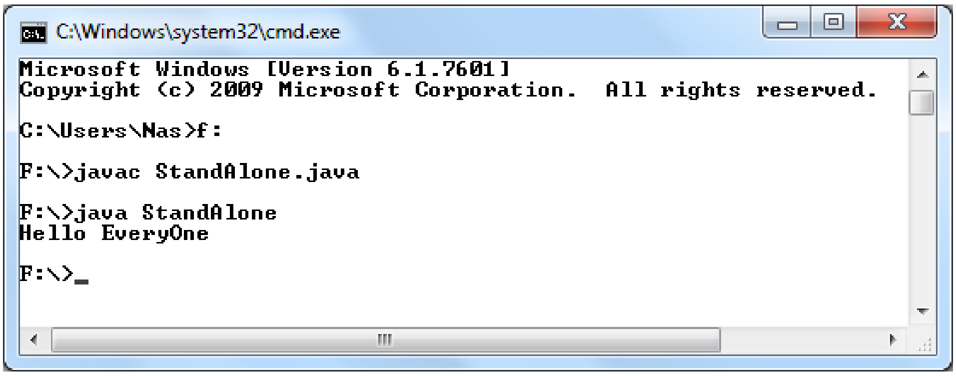
الان قم بترجمة البرنامج (عليك بكتابة اسم البرنامج مع الامتداد) كالتالي :

F:\ javac StandAlone.java

تلاحظ الان ان ملفا بنفس الاسم ولكن بالامتداد (.class) قد تم تكوينه (StandAlone.class) حيث يمكنك استعراض الملفات باستخدام الامر dir ضمن الـdos .

الان قم بتنفيذ البرنامج باستخدام (Java) كالتالي :

فيظهر نتيجة البرنامج وهي طباعة جملة الترحيب كالتالي :



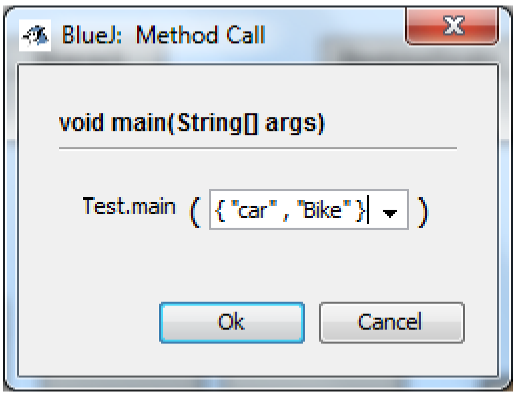
كما ذكرنا سابقا لابد من كتابة دالة main(String[] args) لكي يبدأ تنفيذ البرنامج منها . ولكن ماذا تعني (String[] args) وفي ماذا تستخدم ؟

البراميتر بين الاقواس في منهج main عرف تقليديا كـ args هي مصفوفة من السلاسل الرمزية تستخدم لتمرير قيم للبرنامج عند التنفيذ ، يتم تمرير القيم الى البرنامج وذلك بترك مسافة بعد اسم البرنامج وكتابة القيم والتي سيتم تخزينها في مصفوفة السلاسل الرمزية args[] . فأول قيمة يتم تمريرها الى البرنامج سيتم تخزينها في عنصر المصفوفة ذو الفهرس 0 أي args[0] . وثاني قيمة في العنصر ذو الفهرس 1 أي args[1] ... وهكذا.

فلنفترض ان لدينا برنامج اسمه Test ونريد ان نمرر له كلمة "car" فإننا نكتبها كالتالي :

F:\ java Test car

حيث ان args[0] الان تحتوي على "car". أما من داخل الـBluej فعند النقر بالزر الايمن على الفصيل واختيار الدالة main() فإننا نضيف السلاسل الرمزية بين قوسي الفئة مفصولة عن بعضها بالفواصل ليتم تمريرها الى المصفوفة args كالتالي :



يجب ملاحظة ان القيم الممررة الى البرنامج سيتم تخزينها في **مصفوفة من السلاسل الرمزية** **String** اسمها args ، لذلك اذا اردنا ان نمرر رقم الى البرنامج مثلا وليكن :

F:\java Test 4

فسوف يتم تخزينه كسلسلة رمزية أي "4" وليس كرقم 4 أي ان args[0] الان يحتوي "4" فاذا كانت لدينا جملة الطباعة التالية :

```
System.out.println( args[0] + 5 );
```

فان النتيجة ستكون "45" وليس 9 .

اما اذا اردنا ان نتعامل معها كرقم فيجب تحويلها باستخدام الدالة :

```
System.out.println(Integer.parseInt(args[0]) + 5);
```

الان النتيجة ستكون 9 ، أي قمنا بتحويلها الى رقم أولا ثم جمعناها مع 5 .

مثال :

اكتب برنامج جافا مكون من فصيل اسمه ***Option*** يحتوي على الدالة الرئيسية ***main***() ويتم تمرير سلسلة رمزية اليها هي اختصار لاسم اللغة ومن ثم طباعة اسم اللغة كاملا او طباعة رسالة تبين ان اللغة غير معروفة.

ضمن المفكرة اكتب البرنامج التالي وخزنه على القرص الـF: بالاسم Option.java :

Option.java - Notepad

File Edit Format View Help

```
public class Option
{
    public static void main(String[] args)
    {
        if (args[0].equals("-ar") )
            System.out.println ("Arabic");
        else
            if (args[0].equals("-en") )
                System.out.println ("English");
            else
                if (args[0].equals("-es") )
                    System.out.println ("Espanish");
                else
                    System.out.println ("Unkown Language ... ");
    }
}
```

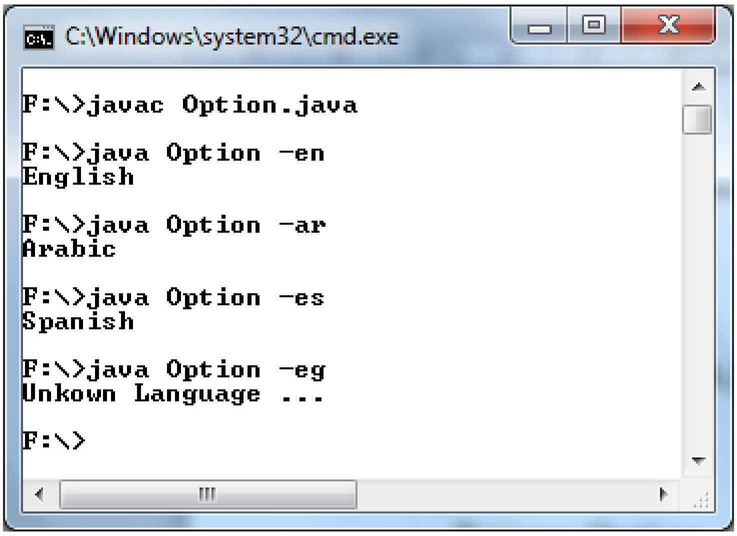
ثم قم بترجمة البرنامج كالتالي :

F:\javac Option.java

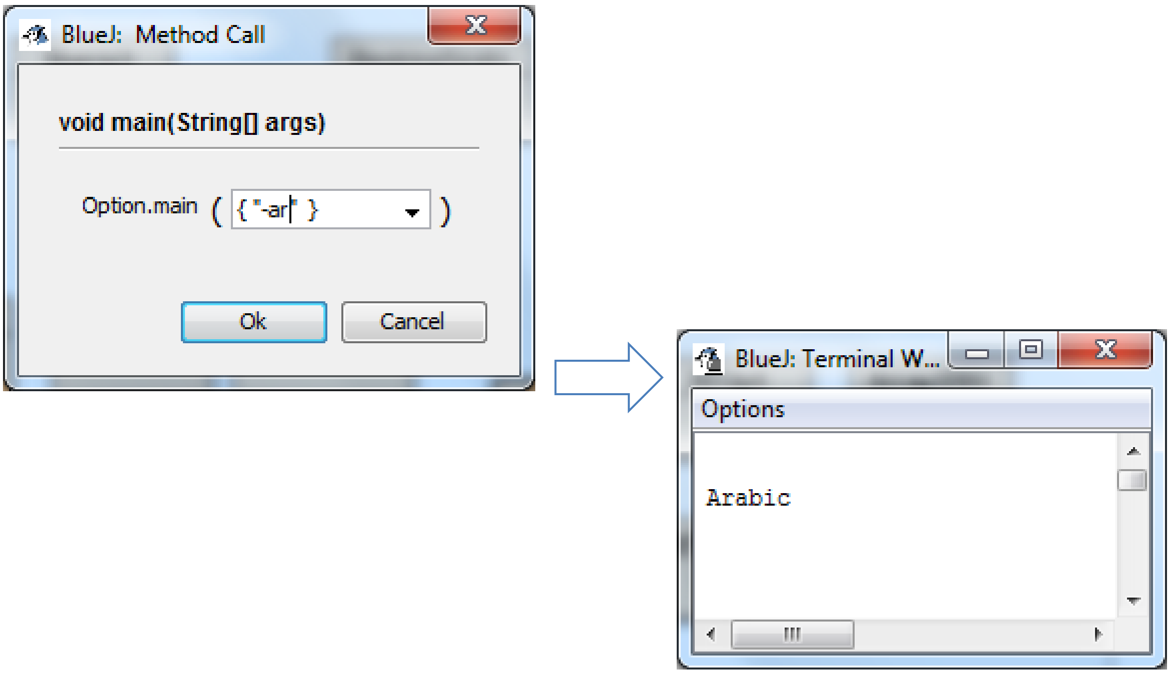
ثم نفذ البرنامج ومرر اليه سلسلة وليكن "-en" مثلا :

F:\java Option –en

فان النتيجة ستكون طباعة كلمة "English" . واليك البرنامج عند تنفيذه بمختلف الخيارات :



اما اذا قمت بكتابة نفس البرنامج ضمن الـBluej وارت تمرير القيمة اليه فيتم ذلك بالنقر على اسم الفصيل ***Option*** واختيار الدالة *main()* ثم ادخال السلسلة الرمزية والتي تمثل الاختصار بين قوسي الفئة كالتالي :



مثال :

اكتب برنامج جافا مكون من فصيل واحد اسمه ***MainMethod*** والذي يقوم باستقبال قيم ممرة كالبارامترات الى الدالة الرئيسية main() تمثل : الاول (رقم) والثاني اشارة حسابية (+ ، - ، x ، /) والثالث (رقم) ، ثم تقوم الدالة الرئيسية main() بإجراء العملية الحسابية المناسبة حسب البارامیتر الثاني او طباعة رسالة تدل على ان الاشارة الحسابية المدخلة كبارامیتر خطأ.

كون ملف نصي جديد باستخدام المفكرة سمه *MainMethod.java* واكتب به الاتي :

```
public class MainMethod
{
    public static void main(String[] args)
    {
          int a,  c ;
          char b;

          a = Integer.parseInt(args[0]);
          c = Integer.parseInt(args[2]);

          b = args[1].charAt(0)  ;

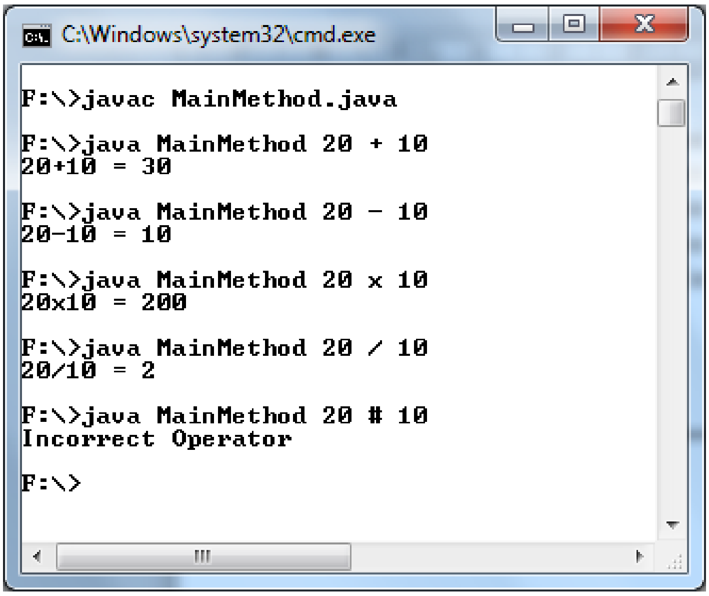
          String msg = args[0]+args[1]+args[2] + " = ";

          switch (b)
          {
            case '+' : System.out.println(msg + (a+c)); break;
            case '-' : System.out.println(msg + (a-c)); break;
            case 'x' : System.out.println(msg + (a*c)); break;
            case '/' : System.out.println(msg + (a/c)); break;
            default : System.out.println("Incorrect Operator");
           }
    }
}
```

تحويل السلاسل الرمزية الى ارقام

تحويل السلاسل الرمزية الى زمر (يمثل الاشارة الحسابية)

قم بترجمة وتنفيذ البرنامج كالتالي :



تدريبات (5)

1- **اكتب برنامج جافا مكون من فصيل اسمه *MaxInArray* يحتوي فقط الدالة الرئيسية *main()* لإدخال سبعة عناصر صحيحة في مصفوفة احادية وطباعة اكبر عنصر في المصفوفة.**

2- **اكتب برنامج جافا مكون من فصيل اسمه *Sorting* يحتوي فقط الدالة الرئيسية *main()* لإدخال خمسة عناصر صحيحة في مصفوفة احادية البعد وطباعة هذه العناصر مرتبة تنازليا.**

3- اكتب برنامج جافا مكون من فصيل اسمه *MyShape* يحتوي فقط الدالة الرئيسية *main()* يستخدم مصفوفة ثنائية البعد 5x5 عند طباعتها ينتج الشكل التالي:

```
* _ _ _ *
_ * _ * _
_ _ * _ _
_ * _ * _
* _ _ _ *
```

4- اكتب برنامج جافا مكون من فصيل اسمه *SeatsBooking* يحتوي فقط على الباني والذي يقوم بفتح رحلة جديدة لحجز الركوب بطائرة مكونة من 10 صفوف واربعة اعمدة من المقاعد، ويحتوي ايضا على منهج اسمه printSeats() يقوم بطباعة مقاعد الطائرة المحجوزة والغير محجوزة ، ايضا يحتوي على منهج اسمه bookSeat() يقوم بحجز مقعد للزبون وطباعة الـok في حالة وجود مقعد فارغ وحسب رغبة الزبون او رسالة تدل على ان المقعد المختار مشغول ولا يمكن حجزه.

5- اكتب برنامج جافا مكون من فصيلين الاول اسمه *MyNumber* (هو اقرب ما يكون الى فصيل Integer من حيث فكرة العمل) بحيث يحتوي على حقل واحد من النوع الصحيح ويحتوي على باني واحد الذي يقوم باستقبال قيمة صحيحة وتخصيصها للحقل ، كما يحتوي على دالة get والذي ترجع قيمة الحقل اضافة الى اعادة برمجة دالة toString() لترجع قيمة الحقل ايضا. يحتوي الفصيل ايضا على اربعة مناهج :

<u>الاول</u> add() والذي يستقبل باراميتر من نوع MyNumber حيث يقوم بجمع قيمة حقل الفصيل مع حقل الباراميتر المستقبل وارجاع كائن من نوع MyNumber تمثل النتيجة

<u>الثاني</u> sub() والذي يستقبل باراميتر من نوع MyNumber حيث يقوم بطرح قيمة حقل الفصيل من حقل الباراميتر المستقبل وارجاع قيمة صحيحة من نوع int تمثل النتيجة.

<u>الثالث</u> multi() والذي يستقبل باراميتر من نوع MyNumber حيث يقوم بضرب قيمة حقل الفصيل مع حقل الباراميتر المستقبل وارجاع قيمة صحيحة من نوع int تمثل النتيجة

<u>الرابع</u> div() والذي يستقبل باراميتر من نوع MyNumber حيث يقوم بقسمة قيمة حقل الفصيل على حقل الباراميتر المستقبل وارجاع قيمة صحيحة من نوع int يمثل النتيجة.

اما الفصيل الثاني فاسمه UseMyNumber والذي يحتوي على الدالة الرئيسية main() تقوم بتكوين كائنين من النوع MyNumber ومن ثم اجراء العمليات الحسابية الاربعة على الكائنين.

إجابة تدريب (5)

1- الفصيل *MaxInArray* لإيجاد اكبر عنصر بالمصفوفة :

```
import java.util.Scanner;

public class MaxInArray

{

    public static void main(String[] args)

    {
          final int MAX = 7;
          int[] a = new int[MAX];

          Scanner kb = new Scanner(System.in);

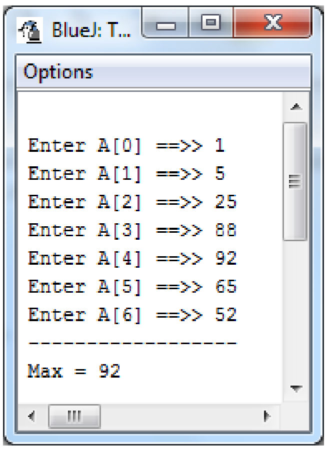
          System.out.println ('\f');

          for(int i = 0 ; i< a.length ; i++)
          {
              System.out.print("Enter A[" + i + "] ==>> " );
              a[i] = kb.nextInt();
          }

          System.out.println("------------------");


          int max = a[0];
          for(int i = 1 ; i< a.length ; i++)
          {
              if (max < a[i])
                  max = a[i];
          }
          System.out.println ("Max = " + max);

    }
}
```



2- الفصيل *Sorting* لترتيب عناصر المصفوفة تنازليا :

```
import java.util.Scanner;

public class Sorting

{
    public static void main(String[] args)
    {
         final int MAX = 5;
         int[] a = new int[MAX];

         Scanner kb = new Scanner(System.in);

         System.out.println ('\f');

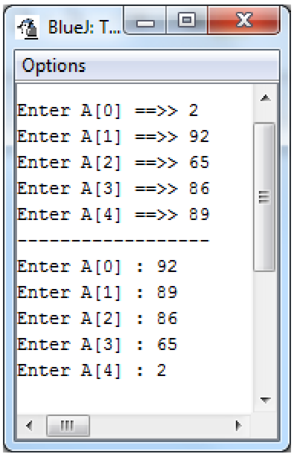
         for(int i = 0 ; i< a.length ; i++)
         {
             System.out.print("Enter A[" + i + "] ==>> " );
             a[i] = kb.nextInt();
         }

         for(int i = 0 ; i< a.length-1 ; i++)
            for (int j = i+1; j < a.length ; j++)
                if (a[i] < a[j])
                {
                     int temp = a[i];
                     a[i] = a[j];
                     a[j] = temp;
                }

          System.out.println("------------------");

          for(int i = 0 ; i< a.length ; i++)
             System.out.println("Enter A["+i+"] : " + a[i]);
    }
}
```

ترتيب عناصر المصفوفة



لترتيب عناصر المصفوفة نستخدم حلقتين متداخلتين: الاولى تبدا من العنصر الاول الى العنصر ما قبل الاخير ، والثانية تبدأ من بعد العنصر المراد اختباره الى اخر عنصر في المصفوفة. فاذا وجدنا اي عنصر يقع بعد العنصر المراد اختباره اكبر منه يتم تبديل العنصرين ليكون الاكبر هو الاول والاصغر بعده.

* اذا اردنا ترتيب عناصر المصفوفة تصاعديا ما علينا الا قلب علامة اصغر من (<) الى علامة اكبر من (>) كالتالي:

```
if (a[i] > a[j])
```

3- الفصيل *MyShape*:

```
import java.util.Scanner;

public class MyShape
{

    public static void main(String[] agrs)
    {
        final int ROWS = 5;
        final int COLS = 5;

        char[][] mat = new char[ROWS][COLS];

        Scanner kb = new Scanner(System.in);

        System.out.print ('\u000C');

        for(int i = 0 ; i< ROWS ; i++)
            for (int j =0 ; j < COLS; j ++)
            {
                if ((i==j) || (j == COLS - i -1 ) )
                    mat[i][j] = '*';
                else
                    mat[i][j] = '-';
            }

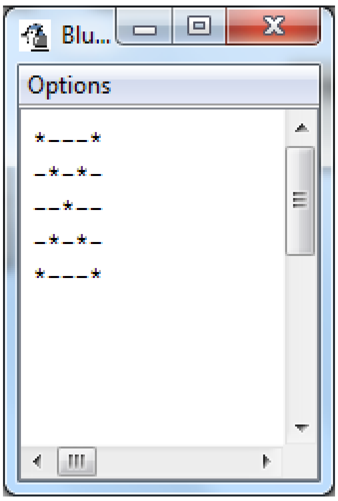
         for(int i = 0 ; i< ROWS ; i++)
         {
             for (int j =0 ; j < COLS; j++)
                  System.out.print ( mat[i][j] );
             System.out.print('\n');
         }

    }

}
```

تحديد عناصر العمود الرئيسي.

تحديد عناصر العمود الثانوي.



4- الفصيل BookingSeats :

```
import java.util.Scanner;
public class SeatsBooking
{
    final int ROWS = 10;
    final int COLS = 4;
    final boolean FREE = true;

    boolean[][] planeSeat = new boolean[ROWS][COLS];

    Scanner kb = new Scanner(System.in);

    public SeatsBooking()
    {
        for(int i = 0 ; i< ROWS ; i++)
            for (int j =0 ; j < COLS; j ++)
                planeSeat[i][j] = FREE;
    }

    public void printSeats()
    {
        System.out.print('\f');

        for(int i = 0 ; i< ROWS ; i++)
        {
            for (int j =0 ; j < COLS; j ++)
            {
                if (planeSeat[i][j] == FREE)
                    System.out.print ("\t-" );
                else
                    System.out.print ("\t#" );
            }
            System.out.print('\n');
        }
    }

    public void bookSeat()
    {
        printSeats();
        System.out.print ("Enter Row & Column : ");

        int i = kb.nextInt();
        int j = kb.nextInt();

        if (planeSeat[i][j] == FREE)
        {
            planeSeat[i][j] = ! FREE;
            System.out.println( "Ok ... ");
        }
        else
            System.out.println ("The Seat is Busy ...");
    }
}
```

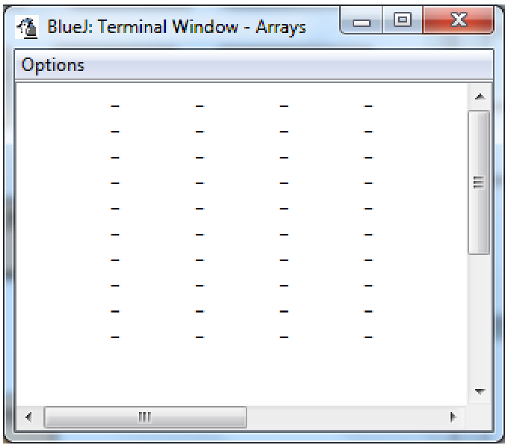
سيتم تمثيل مقاعد الطائرة بمصفوفة ثنائية تتكون من 10 صفوف و 4 اعمدة

فتح رحلة جديدة وذلك بجعل كل عناصر المصفوفة (المقاعد) ليست محجوزة FREE

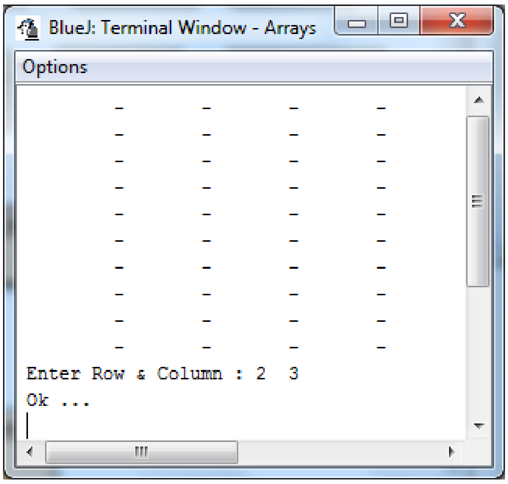
عند طباعة عناصر المصفوفة سيطبع بدل كل مقعد فارغ - ، ومكان كل مقعد محجوز * .

تتم عملية الحجز وذلك بإدخال رقم صف وعمود المقعد ، فاذا كان فارغ ، سيتم حجزه ويطبع ok ، اما اذا كان محجوز تظهر رسالة تفيد ذلك

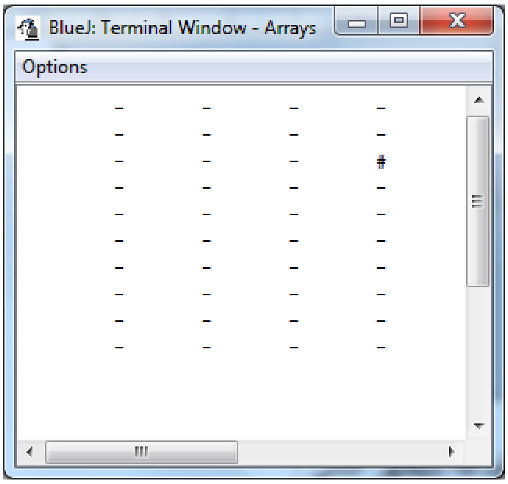
عند تنفيذ البرنامج للمرة الاولى واستدعاء منهج الطباعة سيظهر :



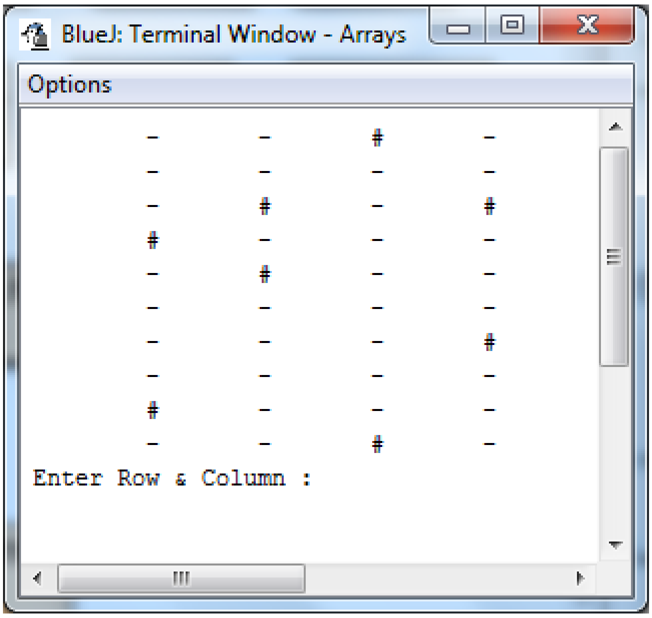
ثم عند استدعاء منهج حجز مقعد وادخال قيم للصف والعمود المراد حجزه وليكن الصف 2 والعمود 3 ستظهر ok لتأكيد الحجز كالتالي :



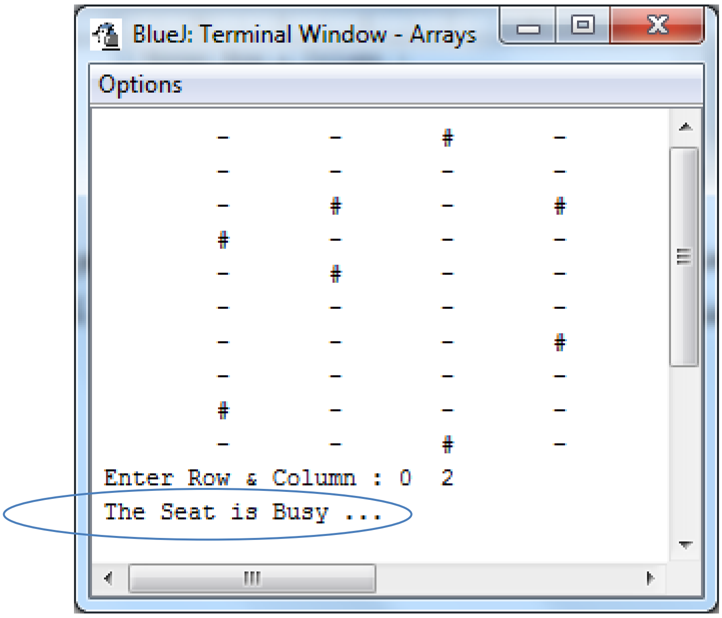
ومن ثم استدعاء منهج الطباعة سيظهر التالي :



وعند تكرار الحجز لعدة مقاعد سيظهر الشكل مشابه للتالي :



اما عند محاولة حجز مقعد تم حجزه مسبقا فان رسالة تفيد بان المقعد محجوز ولا يمكن حجزه ، فمثلا من اخر شكل لو ادخالنا رقم الصف 0 والعمود 2 فان النتيجة ستكون كالتالي :



5- الفصيل Mynumber :

```
public class MyNumber
{
    private int value;
    public MyNumber(int value)
    {
       this.value = value;
    }
    public int getValue()
    {
         return value;
    }
    public String toString()
    {
        return (""+value);
    }
    public MyNumber add(MyNumber n)
    {
         return (new MyNumber(value + n.getValue()) );
    }
    public  int sub(MyNumber n)
    {
         return (value - n.getValue() );
    }
    public int multi(MyNumber n)
    {
         return (value * n.getValue() );
    }
    public int div(MyNumber n)
    {
         return (value / n.getValue() );
    }
}
```

لاحظ أن البارامیتر ھو نفس اسم الفصیل وكذلك القیمة المرجعة ، لذلك كتبنا اسم الفصیل ھنا

لان القیمة المرجعة ھي كائن من نفس نوع اسم الفصیل فاستخدمنا new وكونا كائن جدید من نفس اسم الفصیل بقیمة ھي حاصل جمع قیمة الحقل مع حقل البارامیتر الممر .

لان القیمة المرجعة ھي عدد صحیح int فقمنا بترجیع قیمة صحیحة ھي حاصل ضرب قیمة الحقل مع حقل البارامیتر الممر .

الفصیل UseMynumber :

```
public class UseMyNumber
{
  public static void main(String[] args)
  {
      MyNumber n1 = new MyNumber(20);
      MyNumber n2 = new MyNumber(10);
      System.out.println('\f');
      System.out.println("sum = " + n1.add(n2));
      System.out.println("sub = " + n1.sub(n2));
      System.out.println("Multi = " + n1.multi(n2));
      System.out.println("div = " + n1.div(n2));
  }
}
```

تكوین كائنین من الفصیل السابق

النتیجة المرجعة من استدعاء ھذا المنھج ستكون من نوع الكائن MyNumber ولكن قمنا بإعادة برمجة منھج toString() ، لذلك فان ھذه المنھج سوف یرجع قیمة الكائن value ولیس عنوانه.

اجراء العملیات الحسابیة الثلاثة الباقیة یتم عن طریق استدعاء منھج الكائن ممرا الیه الكائن الاخر لیرجع عدد صحیح

تمرين (3)

1- اكتب برنامج جافا مكون من فصيل اسمه ***AvgMaxMinArray*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال ستة عناصر صحيحة في مصفوفة احادية وطباعة المتوسط الحسابي لأكبر واصغر عنصر بالمصفوفة في المصفوفة.

2- اكتب برنامج جافا مكون من فصيل اسمه **Merging** يحتوي فقط الدالة الرئيسية ***main()*** لدمج عناصر مصفوفتين من العناصر الصحيحة حجم الاولى 4 عناصر والثانية 5 عناصر في مصفوفة واحدة جديدة.

3- اكتب برنامج جافا مكون من فصيل اسمه ***Finding*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال عشرة عناصر صحيحة في مصفوفة احادية البعد وطباعة موقع اول عنصر سالب وكذلك موقع اول عدد زوجي.

4- اكتب برنامج جافا مكون من فصيل اسمه ***Average*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال ثمانية عناصر في مصفوفة احادية البعد وطباعة معدل الارقام الفردية وحاصل جمع الاعداد الزوجية وعدد الارقام السالبة.

5- اكتب برنامج جافا مكون من فصيل اسمه ***SumColumns*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال قيم صحيحة في مصفوفة ثنائية البعد 3 X 3 وطباعتها على شكل مصفوفة ثم طباعة مجموع كل عمود لوحده اسفل المصفوفة.

6- اكتب برنامج جافا مكون من فصيل اسمه ***MatrixOp*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال عناصر صحيحة في مصفوفة ثنائية البعد3 3x وايجاد التالي:

* حاصل ضرب القطر الرئيسي .
* مجموع القطر الثانوي .
* مجموع العمود الاول والعمود الاخير.

7- اكتب برنامج جافا مكون من فصيل اسمه ***PosNegMatrix*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال بيانات صحيحة في مصفوفة حجمها 4x4 وايجاد مجموع القيم الموجبة ومجموع القيم السالبة ومن ثم الفرق بين مجموع القيم الموجبة والقيمة المطلقة لمجموع القيم السالبة.

8- اكتب برنامج جافا مكون من فصيل اسمه ***SumPos2AndMin4*** يحتوي فقط الدالة الرئيسية ***main()*** لإدخال بيانات عددية في مصفوفة ثنائية البعد بحجم (4 X 5) وأوجد مجموع العناصر الموجبة ضمن الصف الثاني، واصغر قيمة في الصف الرابع.